UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E
AUTOMAÇÃO INDUSTRIAL

REDES NEURAIS E LÓGICA FORMAL EM PROCESSAMENTO DE LINGUAGEM NATURAL

Dissertação apresentada à Universidade Estadual de Campinas, como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Aluno: JOÃO LUÍS GARCIA ROSA

Orientador: Prof. Dr. MÁRCIO LUIZ DE ANDRADE NETTO

Campinas, setembro de 1993

AGRADECIMENTOS

Agradeço aos meus pais, Orlando e Helena, sem os quais eu jamais teria chegado até aqui.

Agradeço à minha esposa, Susy, e à minha filha, Mariana, pela compreensão e pela força que me deram durante este longo trabalho.

Agradeço aos colegas do Instituto de Informática da PUCCAMP, especialmente ao seu Diretor, Prof. Otávio Roberto Jacobini, que sempre se mostrou confiante em meu trabalho.

Agradeço, ainda, ao Grupo de Estudos em Inteligência Artificial, do Instituto de Informática da PUCCAMP, que, através de cobranças e colaborações, pôde participar de uma forma direta.

Agradeço ao Prof. Dr. Edson Françozo, do Instituto de Estudos da Linguagem da UNICAMP, que me auxiliou quanto a transcrição correta de termos relacionados a Lingüística.

Agradeço a todos os colegas, alunos e professores, da UNICAMP e do Instituto de Informática da PUCCAMP, que anonimamente, colaboraram de forma direta ou indireta para a realização deste trabalho.

E, finalmente, agradeço ao meu orientador, Prof. Dr. Márcio Luiz de Andrade Netto, que tem me acompanhado durante todo este percurso, acreditando no meu trabalho.

RESUMO

Esta dissertação de mestrado é sobre Processamento de Linguagem Natural (PLN). O PLN consiste de uma série de tarefas que a máquina deve executar para analisar um texto. Na literatura existem vários trabalhos em diversas abordagens. Este trabalho faz uma combinação de abordagens baseadas em lógica e de abordagens conexionistas.

O trabalho proposto tem três partes. A primeira parte faz a análise sintática de frases da língua portuguesa. É baseada em lógica. A segunda parte faz a análise semântica, ou a verificação do significado das palavras numa frase. Isto é feito através de redes neurais artificiais, que "aprendem" a representação binária das palavras (suas microcaracterísticas semânticas). Esta abordagem é chamada de conexionismo. Sua grande vantagem é a habilidade de generalização, ou seja, a rede é capaz de reconhecer uma palavra, mesmo que esta não tenha sido mostrada a ela. A terceira, e última, parte deste trabalho trata da utilização de redes recorrentes para análise de frases. Este tipo de rede serve para "ligar" as palavras em uma frase, pois a rede recorrente tem memória. Ela é capaz de "lembrar" da última palavra vista numa seqüência. É útil para ligar as palavras em uma sentença, por exemplo, o sujeito com o objeto, o objeto com o complemento, etc. Isto torna a frase uma entidade única a ser analisada.

ABSTRACT

This dissertation is about Natural Language Processing (NLP). NLP consists of a series of tasks the machine should carry out in analysing a text. In literature, there are papers having different approaches. This work combines two approaches: based on logic and connectionism.

The proposed work is divided in three parts. The first makes the parsing, or the syntactic analysis of sentences in the Portuguese language, based on logic. The second part takes care of the semantic analysis, or the verification of the meaning of words in a sentence. This is achieved through artificial neural networks that "learn" the binary representation of the words (their semantic microfeatures). This approach is called connectionism. Its major advantage is the ability of generalizing, i. e., it is able to recognize a word even it is not presented to the nets. The third, and last, part of this work is about the use of recurrent networks in text analysis. This kind of network is to "link" the words in a sentence because the recurrent net is given memory, which makes it able to "remember" the last word seen in a sequence. This is useful to link the words in a sentence like the subject to the object, the object to the complement, etc. This makes a sentence an entire item to be analysed.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1. Considerações Iniciais

Existem várias abordagens para o processamento de linguagem natural (PLN). E existem muitos trabalhos publicados nas várias abordagens. Entretanto, combinações das diversas abordagens existentes são raras. Este trabalho ousa em misturar uma abordagem sintática baseada na lógica de predicados, uma abordagem conexionista, baseada em atribuições de casos às palavras, referente à análise semântica, e uma abordagem recorrente, que leva em consideração características temporais da análise da sentença.

A maioria dos trabalhos consultados tratavam de processamento da língua inglesa. Houve necessidade da transposição de procedimentos e idéias para a língua portuguesa, que traz grande dificuldade no tratamento de tempos verbais mas que, em compensação, tem menos palavras ambíguas no seu léxico.

Em relação à ambigüidade, este é o maior desafio enfrentado pelos sistemas que tratam da linguagem natural; identificar o verdadeiro significado de uma determinada palavra pode ser tão complicado que às vezes só é possível através de consulta ao usuário. No sistema proposto, não se tem a intenção de resolver o problema da ambigüidade mas sim contribuir com idéias e apontar direções, com as quais possa-se transpor, ao menos parcialmente, este obstáculo.

O trabalho divide-se basicamente em três etapas, descritas a seguir.

1.1.1. A análise da forma

A primeira trata da **análise sintática** de frases da língua portuguesa, cuja implementação foi baseada na Gramática de Cláusulas Definidas de Pereira e Warren (1980). São frases afirmativas, compostas por até quatro elementos-chaves, que são o sujeito, o verbo, o objeto e o complemento, o qual pode ser o instrumento ou o modificador. Estes elementos-chaves podem vir acompanhados de outras palavras como artigos, adjetivos, partículas reflexivas, etc. A análise sintática faz a verificação da concordância em gênero e número, além de montar uma estrutura, chamada de estrutura-chave, constando apenas dos elementos-chaves, a qual alimentará o analisador semântico (segunda etapa). Por exemplo, a frase:

A menina bonita quebrou a frágil vidraça com um martelo. (1)

gerará a estrutura-chave

menina-quebrar-vidraça-martelo

onde *menina* é o sujeito, *quebrar* é o verbo, *vidraça* é o objeto e *martelo* é o instrumento. Note que uma frase nunca tem, ao mesmo tempo, um instrumento e um modificador. Outra frase pode ser:

Todos os homens comem macarrão com cenouras. (2)

onde *cenouras* é o modificador de *macarrão*. A estrutura-chave da frase anterior será

homem-comer-macarrão-cenoura

É claro que uma frase pode não estar completa. Por exemplo, na frase:

O homem moveu-se. (3)

não há objeto e nem complementos (o verbo *mover* aqui é reflexivo).

O analisador sintático é baseado em regras e foi implementado em Prolog.

1.1.2. A análise do significado

A segunda etapa trata da **análise semântica**. Nesta etapa a frase analisada sintaticamente e estando no formato de estrutura-chave, vai alimentar a entrada de uma rede neural com três camadas de elementos (abordagem **conexionista**), que dirá se a frase tem significado.

Esta etapa foi baseada nos trabalhos de McClelland e Kawamoto (1986) e Waltz e Pollack (1985), que tratam a palavra como um conjunto de microcaracterísticas semânticas. Ou seja, toda palavra é descrita como um vetor de bits, onde cada subconjunto de bits tem um significado, como *humano-não humano*, *frágil-duro*, *masculino-feminino*, etc.

A rede é alimentada com a representação canônica da palavra, ou seja, com o seu conjunto de microcaracterísticas semânticas. Na verdade para um determinado verbo, existem quatro redes: uma para o agente, uma para o paciente, uma para o instrumento e uma para o modificador. Por exemplo, para a frase (1), a rede do agente é ativada para uma estrutura, confronto das micro-características de *menina* com elas mesmas, chamada de **estrutura de sentença**. O formato da saída da rede é chamada de **estrutura de caso**, que é o confronto das microcaracterísticas de *menina* com as do verbo *quebrar*. O processo repete-se para as outras redes.

Como as redes foram treinadas para várias frases, elas têm condição de verificar se uma frase nova, ou seja, uma frase não conhecida, está ou não semanticamente correta. O algorítmo usado para treinar estas redes foi o "**backpropagation**". Este algorítmo consiste basicamente no seguinte. Primeiro, atribui-se pesos aleatórios às ligações entre os elementos das redes. Ao entrar-se com uma estrutura de sentença, a saída da rede é comparada com a saída desejada, ou seja, a saída que deveria ocorrer caso a rede tivesse aprendido aquela estrutura. As ligações são enfra-quecidas ou fortalecidas, para "corrigir" a saída real. Este processo é repetido iterativamente até que a rede apresente erro pequeno, ou seja, até que a rede "aprenda" aquela estrutura.

O sistema conexionista implementado consiste de duas partes. A primeira, a fase do **aprendizado**, consiste na apresentação seqüencial de frases diferentes, mas corretas semanticamente, e na correção de pesos descrita acima onde cada iteração é denominada época. A

segunda parte, a fase do **reconhecimento**, onde é apresentada uma frase nova ou não à rede e ela deve ser capaz de, numa única época, dizer se esta frase está correta semanticamente.

Esta etapa foi implementada na linguagem de programação Pascal.

1.1.3. A análise temporal

A terceira etapa, a **análise recorrente**, trata da verificação das seqüências de palavras previstas. Pode-se, com esta etapa, verificar se as palavras estão numa seqüência apropriada. Esta abordagem foi baseada no trabalho de Elman (1990).

O analisador semântico faz apenas a verificação do elemento para o verbo e do paciente para o modificador, no caso deste existir, ou seja, na frase (2), apenas existe a verificação de relação entre *homem* e *comer*, entre *macarrão* e *comer* e entre *macarrão* e *cenoura*. Não há a verificação entre *homem* e *macarrão* e nem entre *homem* e *cenoura*. Caso deseje-se fazer esta verificação, utiliza-se uma rede recorrente, que é uma rede provida de memória, onde pode-se conhecer uma determinada seqüência de palavras que se ensinou.

O algorítmo empregado nesta rede também foi o "backpropagation". A diferença é que esta rede tem uma camada a mais, onde armazena-se o estado anterior (memória). Na fase do aprendizado, ensina-se à rede todas as seqüências de palavras possíveis para todas as frases possíveis. Na fase de reconhecimento, entra-se com uma palavra e a rede fornece a próxima na seqüência. Esta etapa foi implementada conjuntamente com o analisador semântico, portanto também em Pascal.

Capítulo 2

Conceitos Básicos

2.1. Introdução

O processamento de linguagem natural pode ser definido de formas diferentes. Todas as definições incorporam a noção de armazenamento em computador e manipulação de dados lingüísticos. Entretanto, o ponto de discussão é o grau de sofisticação envolvido, que traduz-se em uma porção de estruturas lingüísticas inerentes ao texto original, as quais o sistema pode detectar automaticamente, armazenar e manipular. De uma forma mais simples, o processamento de linguagem natural pode ser definido como a habilidade de um computador em manipular a mesma linguagem que os humanos usam no dia-a-dia.

O processamento de linguagem natural é geralmente dividido em seis grandes áreas (Obermeier, 1987): (1) interfaces em linguagem natural para bases de dados; (2) tradução de máquina - isto é, de uma linguagem natural para outra; (3) investigação minuciosa de texto, ou programas de indexação inteligentes para sumarização de grandes quantidades de textos; (4) geração de texto para produção automática de documentos padrões; (5) sistemas de fala para permitir interação de voz com computadores; (6) ferramentas para desenvolver sistemas de processamento de linguagem natural para aplicações específicas.

2.2. Conhecimento e modelos

A hipótese de que as pessoas compreendem o mundo através da construção de modelos mentais sugere os itens fundamentais para todos os campos da ciência cognitiva:

- **Psicologia**: Como os modelos são representados no cérebro, como eles interagem com os mecanismos de percepção, memória e aprendizado, e como eles afetam ou controlam o comportamento?

- **Lingüística**: Qual é o relacionamento entre um universo, os objetos que ele nomeia e um modelo mental? Quais são as regras de sintaxe e semântica que relacionam modelos às sentenças?

- **Filosofia**: Qual é o relacionamento entre conhecimento, significado e modelos mentais? Como são os modelos usados no raciocínio e como tal raciocínio está relacionado com a forma lógica?

- **Ciência da computação**: Como um modelo pessoal do mundo pode ser representado em um sistema computacional? Quais as linguagens e ferramentas necessárias para descrever tais modelos e relacioná-los aos sistemas externos? Os modelos podem suportar uma interface de computador que as pessoas achariam simples de usar?

2.3. Itens lingüísticos

2.3.1. Alguns conceitos de lingüística

As **palavras** são as expressões básicas da linguagem. Elas são símbolos que servem para denotar seres, propriedades de seres, relações entre seres, propriedades de relações, etc. As palavras são escritas como cadeias de letras do alfabeto e têm uma estrutura interna de sub-cadeias concatenadas que são chamadas de **morfemas**.

O morfema básico de uma palavra é chamado de **raiz**. A uma raiz pode-se afixar outros morfemas, por exemplo, **prefixos** e **sufixos**.

Cada um destes morfemas tem uma função definida na composição da palavra. O processo de composição é chamado de **análise morfológica**.

2.3.2. Classes de palavras

Substantivos, adjetivos, numerais, pronomes, verbos, advérbios e conectivos são as classes de palavras da língua portuguesa. Os determinantes, normalmente, são os artigos, definidos e indefinidos, mas podem ser numerais, pronomes, etc. Os **sintagmas** são subdivisões das sentenças de uma linguagem natural em que percebe-se um significado claro. Os tipos de sintagmas mais importantes são:

- **Sintagma nominal** (SN): é um trecho da oração que define completamente uma entidade ou conjunto de entidades do mundo do falante.

- **Sintagma verbal** (SV): esta parte da oração define uma atividade ou estado no tempo, como os verbos. Aliás, o verbo é o centro do sintagma verbal. O sintagma verbal afirma algo sobre algum sintagma nominal externo, na maioria das vezes.

- **Sintagma preposicional** (SP): é composto de uma preposição seguida de um sintagma nominal.

- **Sintagma adjetival** (SAdj): ocorre com um verbo de ligação atribuindo qualidades a um sintagma nominal, como em "Paulo é inteligente", ou qualificando um sintagma nominal.

- **Sintagma adverbial** (SAdv): similar ao sintagma adjetival, é composto de um ou mais advérbios seguidos, opcionalmente, por um sintagma preposicional.

- **Oração subordinada adjetiva restritiva** (OSAR): é uma oração onde ocorre um verbo principal, cujo objeto - ou sujeito - está faltando e é referenciado externamente por meio de um SN, SP ou SAdv. Exemplos:

"O gato *que comeu o rato* dormiu."
"O rato *que o gato comeu* morreu."

- **Locução verbal** (LV): é definida simplesmente como um verbo, chamado principal, precedido opcionalmente por verbos auxiliares.

2.3.3. As camadas da linguagem

A **linguagem** é um meio de comunicação. Ela é organizada em um sistema com níveis de regras complexos, dos mais baixos níveis dos sons e ritmos, aos mais altos, do significado e do relacionamento entre linguagem e o mundo. Cada nível trata de um aspecto do processo de comunicação e forma um subsistema completo com seus próprios elementos e regras de combinação. São (Sowa, 1984):

- **prosódia**. Os padrões de ritmo e entonação da linguagem.

- **fonologia**. Os sons, ou fonemas, da linguagem.

- **morfologia**. Os elementos significativos, ou morfemas, que constroem as palavras.

- **sintaxe**. As regras para combinar palavras em frases ou sentenças. A sintaxe estuda as regras da gramática para expressar o significado em uma cadeia de palavras.

- **semântica**. O significado e sua expressão.

- **pragmática**. O uso da linguagem e seus efeitos no ouvinte. A pragmática estuda como o significado básico está relacionado com o contexto corrente e com as expectativas do ouvinte.

2.3.3.1. A sintaxe

A sintaxe é a camada que combina palavras em sentenças. **"Parsing"** (análise) é o ato de aplicar regras gramaticais para determinar como as palavras são combinadas. A **ambigüidade** de

palavras individuais não é o problema mais sério, desde que o contexto seja suficiente para resolver a parte do discurso.

Ambigüidades sintáticas podem ser resolvidas pela semântica. O componente sintático de um analisador percorre a cadeia de entrada e o componente semântico junta grafos canônicos para representar o significado. Quando o componente semântico falha ao achar uma junção satisfatória, executa-se o componente sintático de forma a rejeitar a regra gramatical corrente e tentar uma opção diferente.

As linguagens usam meios sintáticos diferentes para expressar as mesmas relações. Uma linguagem pode expressar certas relações com grande precisão e tolerar ambigüidades em outras.

De todos os níveis da linguagem, a sintaxe é a melhor entendida. A **gramática tradicional** consiste das regras informais que são ensinadas em escolas. A **gramática transformacional** é uma teoria formal, mais detalhada, cujo proponente mais notável é Noam Chomsky (1972).

2.3.3.2. A semântica

A base semântica depende do que o falante sabe sobre o assunto. A forma como o falante apresenta o assunto depende da pragmática - contexto, cirscunstâncias externas e das expectativas do ouvinte. Não há razão para acreditar que todos esses aspectos do significado originam-se de uma estrutura de base simples. Ao invés disto, uma sentença é derivada de seis tipos diferentes de informação:

- **Grafos conceituais** são a forma lógica que estabelece os relacionamentos entre pessoas, coisas, atributos e eventos.

- **Tempo verbal e modalidade** descrevem como os grafos conceituais relacionam-se com o mundo real. Eles estabelecem se alguma coisa aconteceu, pode acontecer, acontecerá ou deveria acontecer.

- **Pressuposição** é a informação de fundo que o falante e o ouvinte assumem.

- **Foco** é o ponto novo que o falante está tentando mostrar.

- **Ligações de correferência** mostram quais conceitos referem-se às mesmas entidades. Numa sentença estas ligações são expressas como pronomes ou outras referências anafóricas.

- **Conotações emocionais** são determinadas pelas associações nas mentes do falante e do ouvinte.

A **percepção** é o processo de construção de um **modelo de trabalho** que representa e interpreta a entrada sensorial. O modelo tem dois componentes: uma parte sensorial formada a partir de um mosaico de elementos chamados "perceptos", cada um dos quais unifica com algum aspecto da entrada; e uma parte mais abstrata chamada de **grafo conceitual**, que descreve como os perceptos juntam-se para formar o mosaico.

Um grafo conceitual é uma combinação de nós de conceito e nós de relação, onde todo arco de toda relacão conceitual é ligado a um conceito. Mas nem todas as combinações fazem sentido. Para distingüir os grafos significativos que representam situações reais ou possíveis no mundo externo, certos grafos são declarados canônicos. Através da experiência, cada pessoa desenvolve uma visão do mundo representada em grafos canônicos.

Como um exemplo, considere a sentença *O homem mordeu um cachorro*. O grafo conceitual da figura 2.1 representa o significado básico desta sentença. O tempo verbal passado e o modo indicativo mostram que o evento realmente ocorreu em algum tempo no passado. A pressuposição é que o sintagma *o homem* é uma referência anafórica a um indivíduo específico no contexto - mesmo que ele tenha acabado de ser mencionado, ou que ele esteja presente na cena. O foco é a nova informação sobre a mordida no cachorro. Desde que *o homem* refira-se a alguma coisa previamente mencionada, uma ligação de correferência conecta o conceito [HOMEM] na figura 2.1 com uma ocorrência prévia de [HOMEM]. Estas cinco primeiras estruturas constituem o significado literal ou conceitual de uma sentença. O sexto, conotação emocional, não é expresso diretamente por palavras numa sentença, nem por conceitos e nem por um grafo. Ele é determinado por associações a uma experiência prévia que depende das pessoas envolvidas.

Figura 2.1. Grafo conceitual para a frase "homem mordendo cachorro".

Os grafos conceituais formam uma linguagem de representação do conhecimento baseada na lingüística, na psicologia e na filosofia. Nos grafos, os nós de conceitos representam entidades, atributos, estados e eventos, e os nós de relação mostram como os conceitos são interconectados.

As regras de formação canônica são **regras de especialização**. A **generalização**, que procede na ordem reversa da especialização, não é necessariamente canônica, mas é importante porque forma a base para a lógica e para a teoria do modelo.

2.3.4. A língua e a linguagem

As línguas não se reduzem a sistemas formais. Sua simples abertura semântica e pragmática, a existência de termos cujo significado é definido pela situação (tais como os demonstrativos e os pronomes pessoais) e as inconsistências da categorização sintática e semântica são suficientes para sustentar esse ponto (Albano, 1989).

A **linguagem** é um sistema de comunicação lingüística, essencialmente vocal (com a sua transcrição gráfica), sendo faculdade, atividade, fenômeno humano global, da **língua**, que por sua vez, é o código do sistema de comunicação, que este utiliza naquela atividade e com aquela faculdade. O fato de uma língua permitir um número infinito de mensagens diferentes e de a gramática que a descreve dever comportar um número finito de regras que permitem produzir um número infinito de frases, desempenhou um papel fundamental na reflexão lingüística contemporânea (Dubois-Charlier, 1976).

2.3.5. A gênese da linguagem

Aprender uma linguagem é difícil, mesmo para um adulto, ou melhor, principalmente para um adulto. Em pouco tempo, as crianças adquirem um conhecimento de sua língua nativa que um adulto, que começa a aprender uma língua estrangeira, nunca consegue.

Para explicar a facilidade da criança para aprender linguagem Chomsky assumiu a existência de um órgão mental dedicado somente à linguagem.

A maturação do cérebro afeta outras funções, da mesma forma que afeta a linguagem. Atividades intelectuais, como música e xadrez, assim como as atividades físicas como ginástica, devem ser aprendidas na infância.

A evidência a partir do aprendizado da linguagem mostra que a linguagem não é um sistema monolítico, mas que consiste de funções diferentes que são adquiridas em estágios diferentes:

- Primeiro a criança associa palavras com os conceitos concretos usados na percepção do mundo e na ação sobre esse mundo;

- Depois ela aprende regras sintáticas para mapear conceitos e relações conceituais para sentenças bem formadas; e

- Finalmente, ela domina as estruturas formais e a sintaxe.

A sintaxe é o estudo dos princípios e dos processos com os quais constroem-se as frases da linguagem. As crianças usam a semântica como um guia para aprender sintaxe. A semântica é o estudo das relações de significado/sentido entre as palavras da frase e das frases entre si. Os adultos, que preocupam-se mais com as estruturas formais, usam a linguagem com grande precisão e eficiência. As relações sintaxe-semântica são importantes para o entendimento humano e também para o processamento no computador.

2.3.6. Gramática gerativa e transformacional

Segundo Dubois-Charlier, "uma teoria lingüística moderna que se atribui explicitamente o objetivo de dar conta de todos os aspectos de uma língua e que, além disto, compreende hipóteses sobre a faculdade da linguagem, sobre a aquisição lingüística e sobre a universalidade da essência dos mecanismos lingüísticos em todas as línguas", chama-se **gramática gerativa e transformacional**.

Nesta gramática é importante a função da frase, que é uma organização, combinação de elementos, agrupados segundo certos princípios que a caracterizam como estrutura. A gramática é um conjunto de regras que permite organizar as palavras de uma linguagem em frases. É um sistema finito de regras que o falante de uma língua interiorizou, na maioria das vezes inconscientemente, e que lhe permite entender e produzir frases dessa língua.

2.3.7. Integrando sintaxe e semântica

Os analisadores que fazem apenas a análise sintática defrontam-se com um número enorme de ambigüidades. Sem a semântica, um analisador não teria como escolher a melhor das análises ambíguas. Sem sintaxe, um programa perderia distinções. Todos os analisadores, mesmo os semânticos, usam alguma sintaxe.

A sintaxe determina que posições as palavras ocupam na sentença e a semântica determina que posições elas ocupam conceitualmente. Os sistemas com semântica mais complexa freqüentemente são tão especializados, que não poderiam ser utilizados para uma outra aplicação, sem terem que ser reescritos completamente.

Ao invés de usar regras sintáticas gerais que relacionam categorias como SN e SV, os sistemas baseados em gramática semântica usam regras orientadas à aplicação que relacionam categorias específicas. A gramática semântica permite acomodar palavras que não estão no dicionário.

Muitos analisadores têm sido projetados para mapear o Inglês em grafos de dependência conceitual. Estes analisadores usam sintaxe. A maioria deles trata a sintaxe como "conhecimento de como combinar significados de palavras baseado nas suas posições na expressão". Alguns usam

as informações sintáticas e semânticas concorrentemente, enquanto constroem uma representação conceitual. Poucos usam ligar a sintaxe e a semântica em "pacotes de palavras" associados com palavras individuais. Ter uma regra ou procedimento separado para cada palavra consome memória e perde generalizações importantes.

Ao invés de um valor de interesse na própria definição da palavra, um **fator de relevância** pode calcular o nível de interesse dinamicamente. O envolvimento emocional, atenção e o contexto podem afetar o nível de interesse. Ter um nível de interesse fixo em cada definição de palavra não é uma solução geral ou flexível.

2.3.7.1. Interações entre os níveis da linguagem

Existem interações entre níveis lingüísticos. Ainda que a fonologia seja um processo de baixo nível, inconsciente, um fonema que é suficientemente estranho, pode exigir uma decisão consciente.

Para alcançar uma velocidade ótima, cada aspecto da linguagem deveria ser interpretado no nível apropriado mais baixo. Os fonemas são interpretados no nível fonológico, e as gramáticas, por regras sintáticas. O que for ambíguo em um nível, entretanto, deve ser resolvido por retroalimentação de um nível mais alto: a sintaxe ajuda a resolver ambigüidades fonológicas, e esquemas conceituais ajudam a resolver ambigüidades sintáticas. Num caso extremo, o conhecimento de fundo do nível mais alto pode ser necessário para resolver uma ambigüidade fonológica num nível mais baixo.

Os conceitos e as percepções são blocos para construção de modelos mentais. Mas regras ou padrões são necessários para organizar os blocos de construção em estruturas maiores. Um "esquema" é uma regra que organiza percepções em uma coisa só.

Segundo Sowa (1984), a linguagem é processada em estágios com interações complexas:

- Fonologia, sintaxe e semântica são independentes, mas interagem entre si.

- Retroalimentação de nível semântico pode, ou encorajar, ou vetar, interpretações nos níveis fonológico ou sintático.

- Um analisador psicologicamente realista requer retroalimentação imediata entre os níveis.

- Com grandes dicionários, um analisador, que empacota toda a informação sintática e semântica com cada palavra, é impraticável. Psicologicamente, ele falha se se considerar a forma com que as pessoas aprendem novas palavras sem ter que modificar sua gramática.

2.3.8. Contexto e conhecimento de fundo

O significado de uma sentença é apenas parcialmente determinado pelo significado de suas palavras. Uma parte grande, se não a maior, do significado vem do contexto, das intenções do falante e das expectativas do ouvinte.

A semântica determina o significado literal. Os outros fatores, que relacionam a linguagem ao mundo, são chamados de **pragmática**. A pragmática é o estudo das relações do usuário da linguagem com a própria linguagem, tendo em vista que este usuário é um interlocutor. Nesta abordagem, terá um papel importante o estudo da **intenção** do locutor e do reconhecimento desta intenção pelo ouvinte.

Os **esquemas** contém o conhecimento de fundo que determina o que é razoável. Como um analisador sintático traduz a sentença de entrada em um grafo conceitual, os mecanismos de inferência juntam esquemas ao grafo e geram preferências e expectativas para combinações prováveis.

Os lingüistas têm especulado que para qualquer fato no universo é possível construir uma sentença gramatical do Inglês que é ambígua para alguém que não conhece o fato. Quando encontrar uma ambigüidade sem solução, o computador pode fazer a mesma coisa que as pessoas fazem - uma pergunta.

O **princípio cooperativo** consiste em adaptar cada contribuição ao propósito ou direção da conversação. Deve-se dizer o essencial, nem de menos nem demais. Tentar fazer da sua contribuição uma verdade. Ser relevante e perspicaz. O princípio cooperativo é uma importante ajuda à comunicação. Ainda que as pessoas não sejam sempre cooperativas. Algumas vezes elas podem estar brincando, serem evasivas, ou mesmo estar mentindo.

Para ser cooperativo, um sistema deve reconhecer e responder às **pressuposições** básicas. O cheque de pressuposições tem sido implementado em muitos sistemas de perguntas em linguagem natural. Quando os termos não estão no dicionário do sistema, ele usa padrões semânticos para determinar o tipo de conceito esperado.

As regras da implicação conversacional requerem que ambos os participantes formem modelos mentais, não apenas do mundo externo, mas também dos modelos que eles supõem que o outro falante está formando.

Um sistema baseado em regras conversacionais forma um modelo de cada falante e procura por um tópico de interesse comum. Ele combina o que foi dito, com o conhecimento de fundo. Além de combinar a informação nova com o conhecimento de fundo, ele forma um modelo do tópico da conversação e o conhecimento do outro falante, desenha inferências plausíveis e introduz informação nova no tempo apropriado.

A **metáfora** é um meio normal de adaptar palavras existentes a novas situações. Usar metáfora é como dar a uma coisa um nome que pertence a alguma outra coisa. A metáfora não é um dispositivo poético obscuro, mas um aspecto fundamental da linguagem que reflete os mecanismos do pensamento.

Os passos para o reconhecimento da metáfora por um computador são (Sowa):

- tentar analisar cada sentença literalmente. Se nenhum grafo canônico puder ser formado para uma sentença gramatical, considere-a como uma possível metáfora.

- checar o catálogo das analogias comuns para achar uma possível transferência de esquemas para um ou mais conceitos na sentença.

- determinar se um grafo conceitual que representa a sentença pode ser canonicamente derivado do esquema transferido.

- se uma interpretação satisfatória for achada, lembrar do esquema que foi transferido, desde que seja esperado ser usado novamente dentro da mesma estória ou conversação.

Um catálogo padrão de metáforas poderia ser usado para interpretar o discurso do dia-a-dia. Quando aprendem a falar, as crianças cometem erros. Mas os seus erros tendem a aparecer de simples desentendimentos, não dos "erros calculados" das metáforas.

2.4. Abordagens do processamento de linguagem natural

O problema central dos sistemas de processamento de linguagem natural é a transformação de uma frase de entrada potencialmente ambígua em uma forma não ambígua que possa ser usada internamente por um sistema de computador. Estas representações internas variam, é claro, de uma aplicação para outra.

A transposição de uma frase potencialmente ambígua para uma representação interna é conhecida como "parsing" (análise). No processamento de linguagem natural, "parsing" é usualmente um processo de combinar os símbolos de uma frase em um grupo que pode ser substituído por um outro símbolo mais geral. Este novo símbolo pode, por sua vez, ser combinado em um outro grupo, e assim por diante, até que uma estrutura permitida apareça.

Cinco tipos diferentes de "parsers" (analisadores) existem: por casamento de padrões, baseado em gramática, semântico, baseado em conhecimento e por redes neurais. Cada um tem uma abordagem única, sem igual, no processamento de linguagem natural.

2.4.1. Abordagens por casamento de padrões

Os primeiros programas de linguagem natural eram baseados na idéia de que os analisadores podiam procurar padrões lingüísticos numa sentença, sem usar qualquer formalismo gramatical explícito. Durante a análise da sentença, o sistema apenas procura por um possível casamento com um número fixo de padrões. Se um casamento é encontrado, o sistema executa determinada ação

(por exemplo, rearranjar a entrada de acordo com um outro padrão). Um exemplo desta abordagem é o sistema psicólogo ELIZA[1].

Os programas por casamento de padrões sem uma base gramatical provaram ser de uso limitado. Eles são úteis apenas se necessita-se de análise parcial ou se outros componentes do sistema podem compensar a perda de informação sintática. O conceito básico de casamento de padrões, entretanto, foi usado nas gramáticas semânticas.

2.4.2. Abordagens baseadas em gramática

O termo "gramática" refere-se a um conjunto de regras que descrevem quais sentenças são parte de uma determinada linguagem. Estas regras são freqüentemente chamadas de regras de reescrita ou produções. Duas regras simples de reescrita são:

$$S \rightarrow SN\ SV$$
$$S \rightarrow SV$$

Estas regras estipulam que uma sentença (S) deve ter um sintagma nominal (SN) e um sintagma verbal (SV), ou apenas um sintagma verbal. Com estas regras pode-se construir uma estrutura em árvore mostrando sem ambigüidade como as palavras se interagem numa sentença (veja figura 2.2). As folhas da árvore são chamadas de palavras terminais. Os outros símbolos, em maiúsculas, são não-terminais. As regras de reescrita geralmente estabelecem como um não-terminal pode ser reescrito como uma cadeia de terminais ou outros não-terminais.

O lingüista Noam Chomsky (1972) dividiu a gramática em quatro tipos, baseado nos tipos de regras que elas usavam. A gramática mais simples, a do tipo 3, é conhecida como **gramática de estado finito** ou **regular**. Ela pode produzir apenas sentenças simples.

---

[1] ELIZA, o programa de casamento de padrões mais conhecido, foi projetado para simular um psicólogo Rogeriano. O programa foi escrito por Joseph Weizenbaum em 1966 e consiste de um conjunto de padrões, onde cada padrão tem um número de respostas associadas. Quando um determinado padrão é encontrado, o programa seleciona uma resposta do conjunto e faz todas as substituições necessárias nesta resposta.

Gramática:

S → SN SV
SN → DET SN1
SN → SN1
SN1 → ADJ SN1
SN1 → SUBST
SV → VERBO ADV
SUBST → programa
VERBO → compila
ADJ → novo
DET → o
ADV → vagarosamente

Sentença:

o novo programa compila vagarosamente

Árvore:

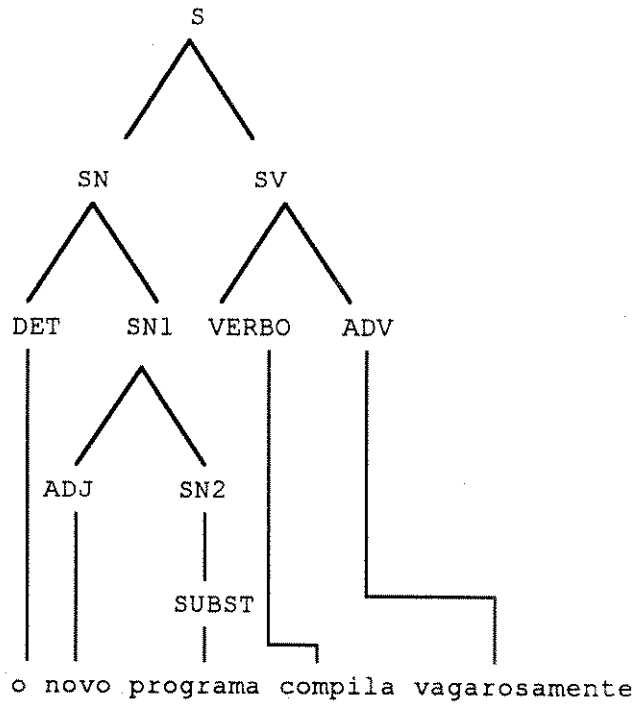


Figura 2.2. Árvore sintática para a sentença "o novo programa compila vagarosamente" (Obermeier, 1987).

A gramática do tipo 2 é conhecida como **gramática livre de contexto**. Nesta gramática, o lado esquerdo de cada regra de reescrita pode consistir de apenas um único símbolo não-terminal. Este símbolo sempre pode ser reescrito como o lado direito da regra, não importando o contexto no qual esse não-terminal aparece.

A outra gramática complexa é a do tipo 1 ou **gramática sensível ao contexto**. Neste tipo, mais de um símbolo pode aparecer no lado esquerdo da regra de reescrita. Existe apenas um requisito para estas regras: devem existir mais símbolos do lado direito que do esquerdo.

A gramática mais complexa, tipo 0, tem regras que não seguem nenhum padrão. Esta gramática é muito difícil de analisar.

Chomsky argumentava que uma linguagem natural como o Inglês, e por conseqüência o Português, não poderia ser completamente descrita por uma gramática livre de contexto, mas por uma gramática sensível ao contexto ou do tipo 0.

Um argumento favorável ao Português ser uma linguagem sensível ao contexto envolve o fato de que substantivos singulares e plurais requerem verbos singulares e plurais, respectivamente. Por exemplo, a gramática da figura 2.2 é uma gramática sensível ao contexto simples. Pode-se expandir esta gramática adicionando três regras de reescrita, todas permitidas:

$$\text{SUBST} \rightarrow \text{programas}$$
$$\text{VERBO} \rightarrow \text{compilam}$$
$$\text{DET} \rightarrow \text{os}$$

Agora entretanto, a gramática permitirá uma sentença que não é válida no Português (falta concordância verbal):

os programas compila vagarosamente.

Este trabalho emprega a abordagem baseada em gramática para construir o analisador sintático.

2.4.3. Abordagens semânticas

O papel central do significado no entendimento da linguagem tem levado os pesquisadores a considerar mais a abordagem semântica do que a sintática. Estes pesquisadores não negam a necessidade de algum processamento estrutural; eles usam a análise sintática para complementar suas considerações semânticas.

Duas abordagens orientadas semanticamente em processamento de linguagem natural são a gramática de caso e a gramática semântica. A idéia da **gramática de caso** é que toda sentença tem uma representação "implícita" de seu significado. Esta representação inclui o verbo e os vários sintagmas nominais relacionados com o verbo. Por exemplo, na sentença "José abriu a porta com a chave" designa-se "José" como o agente, "porta" como o objeto e "chave" como o instrumento para o verbo "abriu". Note que os casos permanecem os mesmos para a sentença "A porta foi aberta por José com a chave". Refere-se aos relacionamentos entre esses substantivos e os verbos como **casos**.

A **gramática semântica** consiste de um léxico e uma série de regras de reescrita. É similar a uma gramática sintática, exceto que classes de palavras (substantivos, verbos, etc.) são substituídas por classes semânticas específicas (navios, propriedades-navios, etc.). A vantagem desta abordagem é que o tamanho destas classes semânticas é muito menor do que o tamanho de uma classe de palavra equivalente. Isto resulta em uma estratégia de análise muito mais eficiente, já que o programa tem que checar um número menor de possibilidades. A desvantagem da gramática semântica é a dificuldade de transferir regras de reescrita de um domínio de aplicação para outro.

2.4.4. Abordagens baseadas em conhecimento

Ao invés de contar apenas com a informação semântica ou estrutural de uma sentença, alguns sistemas de processamento de linguagem natural também têm acesso a uma base de conhecimento para um domínio específico do conhecimento. Isto contrasta com a maioria das teorias baseadas em gramática que enxergam o processamento de linguagem natural simplesmente em termos de um conjunto de regras de reescrita para o processamento a nível da sentença.

Uma abordagem baseada em conhecimento é chamada de **análise especialista em palavra**. Nesta abordagem, a palavra é considerada a unidade lingüística básica. O conhecimento

lingüístico é distribuído entre um grupo de "especialistas" procedimentais que sabem como a interpretação de uma palavra muda em determinados contextos.

Um argumento em favor desta abordagem é o fato de que as palavras têm estruturas lingüística e conceitual ricas. Também é pouco provável que a linguagem possa ser reduzida somente a várias regras de reescrita, como tratam muitas teorias baseadas em gramática.

Uma outra abordagem baseada em conhecimento é chamada de **teoria da dependência conceitual**. A idéia central por trás desta teoria é criar uma representação canônica de uma sentença, baseado em certas primitivas semânticas. Uma representação canônica é simplesmente uma forma básica de representar o significado de uma sentença. Sentenças diferentes que signifiquem a mesma coisa terão a mesma representação canônica. Por exemplo, "Roberto come doce" e "O doce é comido por Roberto" têm a mesma representação canônica:

"Roberto ↔ INGEST ← doce."

Nesta teoria, as primitivas semânticas são as entidades mais básicas usadas para descrever o mundo. Palavras individuais podem sempre ser analisadas de novo, mas primitivas semânticas não.

2.4.5. Abordagem por rede neural

Uma abordagem mais recente em processamento de linguagem natural consiste em estabelecer uma rede de unidades de computação parecidas com o neurônio. Cada unidade tem várias entradas, um conjunto pequeno de estados possíveis e uma saída que é uma função das entradas. Cada entrada para a unidade de computação tem um valor de confiança (peso de conexão), que pode variar de -1 a 1. Quando uma unidade é ativada, ela analisa todas as suas entradas e as pondera de acordo com os seus respectivos valores de confiança. Se certas condições são encontradas, a unidade gera um valor de saída que é usado como entrada por outras unidades. Note que apenas os valores de confiança das entradas podem ser mudados durante o "aprendizado"; o padrão de conexão é estabelecido previamente.

Este tipo de sistema é usualmente chamado de abordagem por rede neural ou **conexionista**. A premissa fundamental desta abordagem é que as unidades individuais não

transmitem grandes quantidades de dados, mas a computação ocorre pois cada unidade é conectada a várias unidades similares.

A análise por rede neural aproxima-se do modelo de processamento de informação lingüística humano, baseado na evidência neurológica.

Este trabalho usa a abordagem por redes neurais para construir os analisadores semântico e recorrente.

2.5. Níveis de análise da linguagem

Os dados estruturais a serem identificados e analisados correspondem aos níveis de análise da linguagem. Como a maioria dos programas tratam apenas de material escrito, o nível fonológico (sistema sonoro) é deixado de lado. Cinco níveis podem ser descritos, segundo Warner (1988), exemplificados pela seguinte descrição de processamento da sentença simples: "O sistema recuperou os documentos".

1. **Morfológico** - palavras (conjuntos de letras separadas por espaço) são decompostas em raízes e terminações. Por exemplo, o termo "documentos" seria quebrado na raiz "documento" e na terminação do plural "s".

2. **Léxico** - usando um dicionário, a cada raiz é atribuído um conjunto de categorias léxicas. Por exemplo, ao termo "documento" deve ser atribuído a categoria léxica SUBSTANTIVO.

3. **Sintático** - usando um módulo de programa chamado de analisador sintático, uma estrutura gramatical é atribuída à sentença. O programa pega o nível de entrada da palavra do componente léxico e decide como as palavras individuais ligam-se para formar sintagmas, cláusulas e sentenças inteiras. Resultaria na seguinte análise da sentença (figura 2.3):

4. **Semântico** - a estrutura sintática é traduzida para alguma forma lógica que representa o significado da sentença que permitirá fazer-se certas inferências. Por exemplo, dada uma representação semântica apropriada, a questão "Os documentos estão no sistema?" poderia ser

respondida - o sistema poderia inferir que recuperação de documentos significa que havia documentos no sistema a serem recuperados.

5. **Pragmático** - este analisa a sentença no contexto, levando em consideração um certo corpo de conhecimento sobre o domínio e sobre os planos e metas do falante e do ouvinte na conversação. Por exemplo, a informação pragmática permitiria ao sistema inferir que um computador foi envolvido na operação de recuperação, ainda que este fato não esteja explícito na sentença.

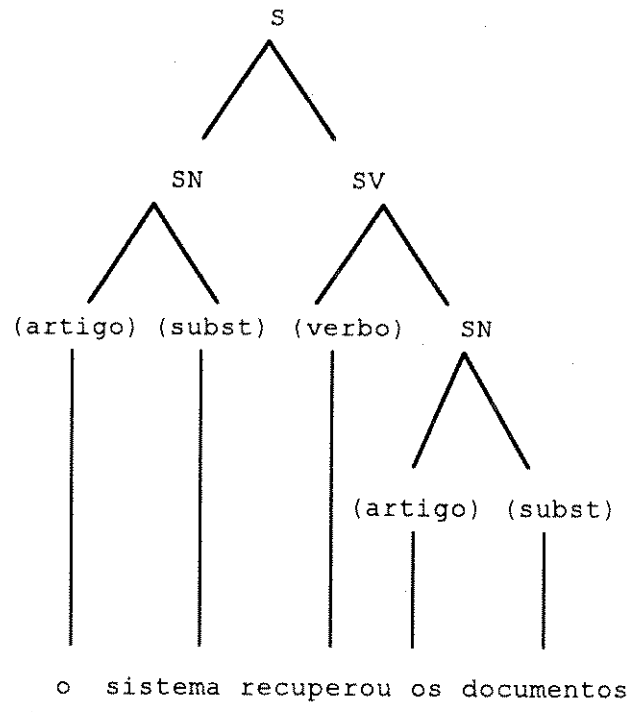


Figura 2.3. Árvore de análise para "O sistema recuperou os documentos" (Warner, 1988).

Os sistemas de processamento de linguagem natural diferem nas quantidades dos cinco tipos de conhecimento que podem explorar e em como o conhecimento é configurado em uma arquitetura de sistemas global.

2.6. Processamento de linguagem natural baseado em lógica

Nos sistemas de processamento de linguagem natural baseados em lógica são descritos os aspectos teóricos e de aplicação. No nível teórico, o sistema é apresentado em **lógica** e no nível de implementação ele é programado em **Prolog**. Este sistema é concebido como um módulo de acesso a uma base de conhecimento. Desta forma, o processamento de linguagem natural é visto como o entendimento de asserções e questões nos limites de um domínio específico.

Em sistemas programados em Prolog, como é o caso do analisador sintático deste sistema, o uso de cláusulas de Horn (um subconjunto da lógica de primeira ordem) como tradução é muito conveniente.

A lógica, usada como linguagem para descrever o mundo do usuário, também deverá ter uma semântica de condições de verdade e modelos. A tradução de uma frase da linguagem natural para uma fórmula da lógica deve ser tal que o valor semântico de ambas seja o mesmo, relativo a um certo modelo.

2.6.1. Problemas de ambigüidade

A ambigüidade pode ser **léxica** quando a mesma palavra, ou expressão básica, denota entidades diferentes no mundo. Normalmente, a ambigüidade léxica é resolvida pelo contexto da frase. Na ambigüidade **sintática**, pode-se ter mais de uma árvore sintática para derivar a mesma frase. Pode acontecer em frases com mais de um verbo, em que não está claro qual é o verbo principal:

Pedro viu Maria passeando.

Em alguns casos, a análise semântica pode resolver ambigüidades sintáticas. Pode-se ter também uma ambigüidade sintática, provocada pela forma das regras, em uma frase com um só significado semântico. Isto é, tem-se várias árvores sintáticas mas um só significado.

A ambigüidade **semântica** é o caso em que tem-se mais de um significado para uma frase. Ocorre acompanhada da ambigüidade sintática, quando as diversas árvores sintáticas produzem análises semânticas válidas, como na frase anterior.

2.7. Técnicas de análise

Existem dois grupos de técnicas de análise: não-determinística e determinística. As não-determinísticas podem ser divididas em analisadores "top-down" e "bottom-up".

Os analisadores **"top-down"** tentam casar as regras da gramática com a entrada, começando na regra de reescrita mais alta (que usualmente envolve o símbolo inicial ou símbolo da sentença S) e recursivamente move-se em direção à mais baixa, a regra de reescrita mais específica. O analisador é bem sucedido se uma sentença pode ser construída a partir da sentença de entrada (casando).

Os analisadores "top-down" são fáceis de escrever e de modificar. Regras que são mais usadas podem ser colocadas na frente de regras menos usadas, aumentando a performance. E o número de sentenças geradas pode ser limitado arbitrariamente.

Contudo, os analisadores "top-down" podem ser lentos. Se todas as regras de um nível falham, o analisador volta ("backtracks") para o nível anterior para tentar uma outra regra lá. Durante o "backtracking", os mesmos componentes podem ser analisados muitas vezes. Os analisadores "top-down" também têm problemas para manipulação de entrada disforme e requerem um módulo separado para decidir qual das muitas análises bem sucedidas é a melhor.

Os analisadores **"bottom-up"** iniciam combinando os elementos do nível mais baixo, e então constroem para cima componentes maiores. Por exemplo, na figura 2.3, os primeiros passos de um analisador "bottom-up" seriam: substituir o não-terminal DET por "o"; ADJ por "novo"; SUBST por "programa"; e SN1 por ADJ SUBST.

Os analisadores "bottom-up" podem, ao menos parcialmente, analisar entrada disforme. E alguns mecanismos podem ser aplicados para reduzir a explosão combinatorial de possíveis

análises. Como os analisadores "bottom-up" não são direcionados à meta, eles geram numerosas análises espúrias. E a correção da análise final só pode ser determinada depois que todas as análises possíveis forem realizadas.

A análise **determinística** não tem "backtracking". É também chamada de análise "espere-e-veja". A técnica cria novos nós de um modo "bottom-up" mas usa uma característica limitada de "olhar-a-frente" para determinar qual nó usar. Uma vantagem do determinístico é o aumento da velocidade porque ele evita a explosão combinatorial para possíveis análises. A desvantagem é que o algorítmo é baseado apenas na informação sintática.

Os analisadores "bottom-up" começam a crescer suas árvores de análise dos constituintes de baixo nível (palavras individuais) até os níveis mais altos, como SN, SV, e finalmente, S. Num analisador "top-down", é dado o controle primeiro a uma meta de alto nível como S, e ela determina quais submetas tentar a seguir. Para manipular o não-determinismo, os analisadores "top-down" usam "backtracking". Os analisadores "bottom-up" freqüentemente aplicam todas as regras em paralelo.

2.8. Redes de Transição Aumentadas

2.8.1. Redes de transição

Uma **rede de transição** consiste de uma série de estados conectados por arcos. Cada arco é rotulado por uma categoria de palavra (ex. substantivo ou verbo) ou uma palavra específica. O programa começa num dado estado e então checa a próxima palavra na cadeia de entrada com o propósito de "casar" com um dos arcos. Se um casamento ocorre, o programa vai para o próximo arco e "atravessa" a rede.

A vantagem das redes de transição é que elas podem ser facilmente implementadas em um computador. Cada estado pode ser implementado como uma função que checa suas entradas com os arcos daquele estado. Se um casamento é encontrado, a função (ou estado) no final deste arco é chamada.

SN simples:

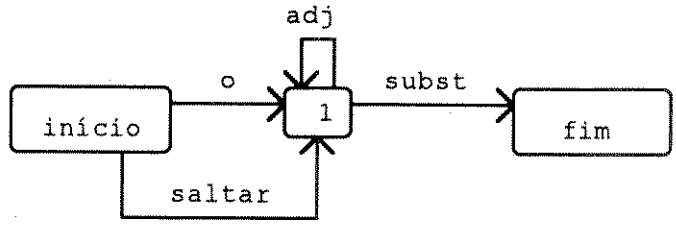
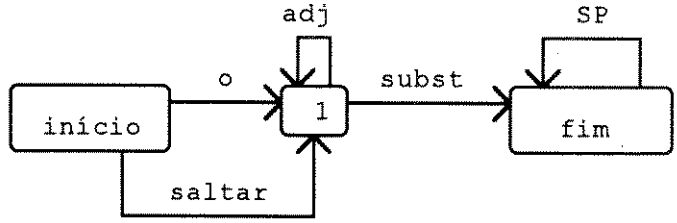


Figura 2.4. Rede de Transição.

2.8.2. Redes de transição recursivas

As redes de transição recursivas têm uma característica adicional na qual alguns arcos podem ser rotulados com outra rede de transição subordinada:

SN:



SP:

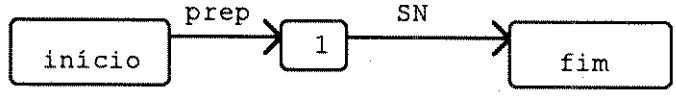


Figura 2.5. Redes de Transição Recursivas.

Estes arcos permitem que uma dada rede chame uma outra ou mesmo que se chame recursivamente. Usualmente, estes arcos são rotulados com símbolos não-terminais (ex. SP para

sintagma preposicional). Estas redes são consideravelmente mais poderosas que as redes de transição.

2.8.3. Redes de transição aumentadas

Para a manipulação de linguagens naturais, vários pesquisadores em Inteligência Artificial estenderam e refinaram as redes de transição para as **redes de transição aumentadas** ("Augmented Transition Networks" - ATNs), que realizam testes e operações de estrutura de construção durante a análise. As ATNs são similares às redes de transição recursivas mas têm três características adicionais: registros, que podem armazenar condições ou informação sobre uma base global; condições, que permitem que arcos sejam selecionados se os registros indicam certas condições; e ações, que permitem que arcos modifiquem a estrutura dos dados. A idéia básica é traduzir uma gramática em uma coleção de máquinas de estado finito, ou redes de transição de estado, que podem chamar uma à outra, recursivamente.

Uma vantagem prática das ATNs é sua notação gráfica para gramáticas, mas essa notação não tem importância teórica. As ATNs têm duas notações: uma é a forma gráfica baseada em coleções de máquinas de estado finito e a outra é a forma executável.

Note que os dados em uma ATN podem ser rotulados, não apenas com palavras, classes de palavras e não-terminais, mas também com testes arbitrários que dependem do estado dos registros globais. Estes registros globais e seus testes associados tornam possível para o programa checar apenas elementos adjacentes. Óbviamente, sendo acapaz de armazenar e agir nestas condições, torna-se possível uma análise muito mais eficiente do que as redes de transição anteriores faziam. Por exemplo, a presença de uma forma do verbo *ser* antes do verbo principal numa sentença pode disparar o cheque para a preposição *por*, o que por sua vez pode confirmar evidência para uma sentença na voz ativa ou passiva.

Ainda que as ATNs sejam poderosas, elas encontram problemas com sentenças gramaticalmente inválidas, para as quais não existem redes. Se o programa encontra uma construção para a qual não tem uma descrição estrutural, ele simplesmente pára.

Como as redes de transição podem ser aumentadas com operações para construção e teste de estruturas, a gramática de estrutura de frase ordinária pode ser estendida a **gramática de**

estrutura de frase aumentada (APSG). As gramáticas de metamorfose baseadas em Prolog são equivalentes às APSGs, que são executadas num modo "top-down".

Ainda que as APSGs e as ATNs sejam dirigidas a regras, suas regras podem ser vistas como programas miniaturas escritos em uma linguagem altamente especializada.

2.9. A Representação da linguagem natural

Para usar linguagem natural nas interfaces homem-computador, deve-se primeiro construir uma estrutura compreensiva para representar sentenças em linguagem natural. Usando Prolog pode-se construir uma estrutura de lista sintática e uma estrutura sintático-semântica; pode-se usar Prolog também para incorporar a estrutura semântica em uma estrutura de fundo que transporte os significados de determinadas palavras da sentença no contexto do conhecimento do mundo geral. Esta estrutura semântica é então colocada no contexto da sentença a ser analisada. Pode-se usar esta técnica de entendimento da linguagem natural no desenvolvimento de interfaces amigáveis.

Pode-se usar a gramática de cláusulas definidas, que é um método de expressar regras livres de contexto como comandos lógicos de um tipo restrito, na análise sintática de sentenças em linguagem natural. Este método tem muitas vantagens, incluindo suporte para dependência de contexto, habilidade de permitir que estruturas sejam construídas durante a análise e habilidade de permitir que condições extras sejam especificadas nas regras gramaticais.

2.9.1. Interpretação da linguagem

Os principais aspectos na habilidade de um computador em entender uma sentença em linguagem natural são a análise sintática e a adição de características às palavras, a alocação de funções às palavras e a fixação da sentença na sua estrutura de contexto. A adição de características novas às palavras significa a incorporação ao sistemas de entendimento interativo características como número e gênero. A alocação de funções às palavras significa explicitamente a expressão

das funções executadas por cada palavra na sentença; estas funções podem ser um agente ou um objeto.

As pessoas e os computadores devem trabalhar com a sintaxe e a semântica cooperativamente, mas existem muitas formas nas quais sintaxe e semântica podem interagir. A análise pode produzir uma representação sintática completa que é então interpretada semanticamente. Usando uma fase de análise separada para concisamente representar os relacionamentos sintáticos é uma vantagem porque muitas regras semânticas diferentes pedem essencialmente a mesma informação sintática.

2.9.2. A adição de características às palavras

Para checar a compatibilidade de número e gênero, consistência de tipos de verbos e certas inconsistências semânticas, deve-se adicionar informação ao analisador sobre palavras definindo suas características. Com esta informação, o analisador pode fazer um cheque de consistência para eliminar estruturas incorretas em sintaxe e em semântica (Geetha e Subramanian, 1990):

. *Compatibilidade de número*. Considere a sentença "Ele gostam de estudar". O número do substantivo e do verbo não casam e então a sentença está sintaticamente incorreta. A incompatibilidade está entre palavras pertencentes a sintagmas sintáticos diferentes, que são os grupos formados por palavras da sentença baseados nas suas categorias sintáticas (como sintagma nominal e sintagma verbal).

Considere um outro exemplo: "Maria comeu estas manga". Neste caso, a incompatibilidade está entre palavras pertencentes ao mesmo sintagma sintático. Isto traz a necessidade de adicionar o número como uma característica da palavra.

. *Compatibilidade de gênero*. Considere a sentença "José e o amigo dela foram ao parque". Quando considerada isolada, esta sentença mostra que os gêneros de "José" e "dela" não casam, então a conclusão é que o amigo não é de *José*. Para achar a incompatibilidade, é necessário conhecer os gêneros de "José" e "dela". Então, deve-se adicionar o gênero como uma característica da palavra.

. *Consistência de tipo de verbos*. Os verbos têm muitas características que ditam a estrutura do resto da sentença. Considere a sentença "Henrique dormiu Tom". "Dormir" é um verbo intransitivo, que significa que ele não age num objeto específico, então ele não pode ter um SN como objeto direto. No máximo, sintagmas preposicionais ou adverbiais como "com Tom" ou "ruidosamente". Portanto, o analisador deve ter a informação de que o verbo é transitivo ou intransitivo.

. *Inconsistências semânticas*. Número, gênero e tipo de verbo são usados apenas para checar o aspecto sintático das sentenças. Pode-se usar certas características de palavras para achar as inconsistências semânticas que dependam somente da natureza de determinadas palavras. Considere a sentença "Ivan come pessoas". O analisador checa as características do sujeito ("Ivan") e objeto ("pessoas") para ver se eles são semanticamente compatíveis com o verbo ("come"). Neste exemplo, o verbo requer um sujeito animado e um objeto não humano, então existe uma inconsistência semântica mesmo que a sentença esteja sintaticamente correta. (Para as exceções, como *Ivan* ser um canibal, adicionar-se-ia este fato ao sistema, que daria precedência sobre as regras gerais no léxico).

2.9.3. A estrutura léxica

O léxico que o analisador usa para sua análise em linguagem natural deve ter dois tipos de informação: a informação sintática e os detalhes semânticos. Ele usa esta informação para auxiliar a análise sintática e mais tarde construir a representação contextual e semântica. A forma da entrada léxica é diferente para categorias sintáticas diferentes.

Substantivos, adjetivos, advérbios, e outras partes do discurso - exceto verbos - têm a mesma estrutura no léxico:

tipo_sintático da palavra (X, R, N, G, T)

O tipo_sintático da palavra seria um "token" como é_substantivo, é_pronome, ou é_nome_próprio. X é a palavra do texto de entrada, R é a raiz da palavra, N é o número da palavra (singular, plural ou indeterminado), G é o gênero (masculino ou feminino) e T é uma lista contendo propriedades semânticas como humano, animado ou entidade.

Para verbos, o léxico também inclui características do verbo como tipo (transitivo ou intransitivo) para análise semântica; estas características podem ser incluídas como regras separadas. Pode-se usar outras características (como pessoa e tempo verbal) diretamente (por casamento de padrões simples) para achar anomalias sintáticas como discordância de sujeito-verbo (por exemplo, "Ele comem"). A estrutura para verbos do léxico é:

é_verbo (X, R, P, TV, T)

onde X é a palavra do texto de entrada, R é a raiz do verbo, P é a pessoa, TV é o tempo verbal (como passado simples, presente, futuro, etc.) e T é o tipo (transitivo ou intransitivo).

Depois da análise sintática, o analisador agrupa as palavras em sintagmas sintáticos como SN, SV e SP. Cada sintagma sintático tem uma palavra principal da qual o sintagma depende e o sintagma sintático aceita as características desta palavra principal. O sistema determina quais funções estes sintagmas executam, aplicando regras apropriadas e casamento de padrões.

2.9.4. A representação semântica

As funções reais, as quais definem as relações que os outros sintagmas sintáticos têm com o verbo, são executadas por sintagmas sintáticos operando como uma unidade. O conhecimento das diferentes funções executadas por estes sintagmas dá ao computador uma compreensão parcial do significado da sentença. O significado da estrutura inteira depende do verbo e das outras unidades que executam funções relacionadas com o verbo principal. Cada verbo terá uma estrutura que será armazenada no léxico. Outros constituintes sintáticos serão alocados às suas respectivas posições se eles satisfizerem os limites semânticos dados.

2.10. Lógica e Linguagem Natural

Lógica é o ramo do conhecimento que trata da verdade e da inferência - ou seja, como determinar as condições sob as quais uma proposição possa ser verdadeira, ou possa ser inferida a

partir de outras proposições. Tal conhecimento é essencial para a comunicação pois a maioria das crenças humanas sobre o universo vêm, não diretamente do contato com ele, mas do que as pessoas contam às outras.

Várias pesquisas em interações de linguagem natural com máquinas usam a lógica como base, incluindo análise, interpretação semântica e raciocínio. Entretanto, não apenas lógica de primeira ordem, mas também formas de lógica mais poderosas - ou pelo menos, muito diferentes - estão sendo usadas no entendimento e geração de linguagem natural.

A idéia de se usar lógica como um suporte conceitual em sistemas de perguntas e respostas não é nova. O fato de que se pode formalmente tratar com a noção de conseqüência lógica torna-a atrativa para a representação do significado. O cálculo de predicados padrão, entretanto, não parece adequado para representar todas as características semânticas da linguagem natural, por exemplo, pressuposições e as sutilezas do significado envolvidas nos quantificadores da linguagem natural. Entretanto, alguns desenvolvimentos indicam que a lógica tem um papel importante no processamento de linguagem natural (Dahl, 1981).

Em primeiro lugar, a pesquisa lingüística chegou a resultados interessantes considerando a extensão do cálculo de predicados padrão a fim de prover um melhor modelo de linguagem formal.

Em segundo lugar, a programação em lógica tornou-se possível desde o desenvolvimento da linguagem de programação Prolog. A lógica pode agora ser usada tanto como um formalismo básico quanto como a ferramenta de programação.

Em terceiro lugar, muitas das implementações em Prolog incluem uma versão da gramática de metamorfose (GM), um formalismo baseado em lógica útil para descrever processadores de linguagem natural em termos de regras de reescrita muito mais poderosas.

Finalmente, a evolução da tecnologia de base de dados tem se voltado cada vez mais na direção do uso da lógica, seja na descrição dos dados, seja para perguntas.

O entendimento e a geração da linguagem são exercícios do raciocínio. Para o raciocínio, a lógica é a melhor ferramenta.

2.10.1. O que é lógica?

Definir lógica não é uma tarefa simples, mesmo se se considerar apenas sistemas formais desenvolvidos pelos lógicos matemáticos. Estes sistemas foram originalmente criados e estudados com a preocupação de caracterizar uma linguagem simbólica dentro da qual todas as proposições matemáticas possam ser expressas. Mais particularmente, os lógicos queriam uma linguagem dentro da qual eles poderiam expressar um conjunto de verdades matemáticas básicas, ou axiomas, a partir dos quais todo o resto poderia ser gerado, aplicando um conjunto finito de regras de prova caracterizadas (combinatoriais), que poderiam ser mostradas preservadoras da verdade.

Em resumo, as linguagens da lógica matemática não serviam para o uso geral. Seus desenvolvedores não defendem que elas sejam simbolismos universais para aplicações irrestritas - ou seja, que tudo pensável possa ser adequadamente expresso nelas. Certamente, nem tudo que é expresso em linguagem natural pode ser expresso em uma linguagem lógica formalizada.

Alguns esforços têm sido feitos nesta direção. Dúvidas têm sido levantadas sobre a adequação ou apropriação, por exemplo, da linguagem "básica" da lógica - a linguagem do cálculo de predicados de primeira ordem.

Uma linguagem formal é determinada pela especificação de um vocabulário, constituído de tipos sintáticos, e pelo desenvolvimento de um conjunto de regras de formação para geração de expressões complexas, em particular, termos e/ou fórmulas complexas bem formadas da linguagem. Todas essas especificações devem ser mecânicas ou algorítmicas.

Tendo determinado a sintaxe de uma linguagem, pode-se então especificar a semântica de uma forma que reflita a especificação recursiva de sua sintaxe. Pode-se então atribuir valores semânticos às expressões primitivas, não lógicas, da linguagem, com tipos diferentes de valor semântico, sendo associados com tipos sintáticos diferentes. Tais atribuições também são chamadas de modelos ou interpretações. Pode-se então dar um conjunto de regras de interpretação que determina valores semânticos de expressões complexas, como uma função de valores semânticos de seus constituintes e das regras de formação sintática usadas para gerá-los. Por exemplo, o significado de constantes lógicas - conectivos funcionais veritativos e quantificadores - pode ser dado pelas regras de interpretação que atribuem valores a expressões complexas que as contêm.

Finalmente, pode-se especificar um aparato dedutivo para a linguagem. Este aparato pode tomar várias formas, mas todas envolvem a especificação de um conjunto de regras de transformação, cuja aplicabilidade para um conjunto de sentenças pode ser efetivamente determinado, e cuja saída, tipicamente uma sentença, pode também ser determinada. Adicionalmente, estas regras devem consistir com a semântica especificada para a linguagem. Por exemplo, se as premissas de uma regra são válidas - verdadeiras em toda interpretação, de acordo com a semântica - então a conclusão da regra também é. Tais regras são ditas consistentes com respeito a validade. Deve-se também requerer que as regras sejam preservadoras da verdade.

2.10.2. Raciocínio e lógica

Para conseguir a liberdade que se tem na escolha de regras, deve-se claramente distingüir raciocínio de prova. O raciocínio ideal pode, freqüentemente, conduzir de crenças verdadeiras para falsas. O raciocínio freqüentemente faz com que se desista de algumas crenças, a partir das quais se inicia - mesmo quando não se resolve intencionalmente colocar essas crenças em teste (em contraste às provas por refutação, ou por redução ao absurdo, na lógica).

Para considerar um caso simples, suponha que aceite-se - entre outras coisas, é claro - alguma sentença na forma "se **P**, então **Q**" e aceite-se o antecedente. Deveria-se aceitar o conseqüente? A resposta é "não necessariamente", pois imagine que se tenha razões suficientemente fortes para acreditar em não-**Q**, e isto levaria a desistência das crenças no condicional ou seu antecedente. Entretanto, as regras de prova são locais; elas aplicam-se a um dado conjunto de sentenças de acordo com as suas formas sintáticas individuais. O raciocínio, por outro lado, pode freqüentemente ser global; deve-se tentar não apenas levar em consideração todas as evidências relevantes, mas também conseguir mais evidências, se a evidência imediata é julgada insuficiente. Estes julgamentos sobre a relevância e pesos de evidência são tipicamente os produtos do raciocínio.

Pode parecer que a prova lógica está sendo colocada em oposição ao raciocínio. A visão correta é que a prova lógica é uma ferramenta usada no raciocínio. Portanto, falar em "raciocínio lógico" não é apropriado.

Os esforços para resolver o problema da representação do conhecimento envolvem dois grandes obstáculos: decidir que conhecimento representar e obter respostas de um computador num tempo razoável.

2.10.3. Lógica "default"

A noção de "pressuposicão" vem da lingüística. Em termos pragmáticos, as pressuposições de uma asserção, uma pergunta ou um comando são aquelas proposições que um falante assume como válidas para o seu universo.

Considere o verbo "lamentar" em duas sentenças, cujos objetos são cláusulas relativas (ou seja, proposições embutidas):

A. José lamenta que Maria tenha vindo à festa.
B. José não lamenta que Maria tenha vindo à festa.

Como todos os verbos "factuais" (que representam um fato), dada nenhuma informação ao contrário, entende-se que o falante assumiu a proposicão embutida - neste caso, que Maria veio à festa. Por outro lado, na sentença

C. José não lamenta que Maria tenha vindo à festa, porque ela não veio à festa.

o falante asserta através da cláusula "porque" que Maria não veio à festa. Então não é consistente acreditar que o falante assuma que Maria veio à festa e a pressuposição não seja gerada.

2.10.4. Lógica modal para planejamento de expressão

A lógica e a dedução têm importantes papéis na geração da linguagem natural. Por exemplo, suponha que haja alguma coisa que uma determinada pessoa A não possa fazer mas uma outra pessoa B pode e faria para a pessoa A, se soubesse do que se trata. O que a pessoa A pode fazer é formular uma pergunta que informe à pessoa B o que deve ser feito e como fazê-lo. O sucesso de uma expressão, emitida pela pessoa A, tal como "a pessoa B poderia pegar aquela chave inglesa na caixa de ferramentas próxima ao martelo?" vem de (1) direcionando a atenção da pessoa B para a coisa na caixa de ferramentas próxima ao martelo, (2) informando à pessoa B o que é uma chave inglesa (se ela ainda não souber) e (3) pedindo à pessoa B que a pegue para a pessoa A.

A produção de uma expressão bem sucedida pode requerer raciocínio sobre o que o agente sabe ou não e o que ele deve fazer para saber. Entretanto, o raciocínio sobre o conhecimento, ação e o efeito da ação no conhecimento, não está dentro do escopo da lógica de primeira ordem, assumindo sua semântica padrão, parcialmente porque os predicados de "conhecimento" são obscuros. Ao contrário dos predicados padrões, os predicados de "conhecimento" não permitem a substituição de termos equivalentes, tal como, se CORRE(Pedro, 2milhas) é verdadeiro, então CORRE(Pedro, 1.24kms) também é. Mesmo que SABE(Pedro, 2milhas = 2milhas) seja verdadeiro, o comando SABE(Pedro, 2milhas = 1.24kms) pode não ser, porque Pedro pode não conhecer nada sobre conversão métrica. O problema é que a ação muda o universo e o que é sabido sobre ele, falsificando coisas que eram verdadeiras e vice-versa. A lógica de primeira ordem, com sua semântica padrão, utiliza verdades eternas.

A lógica modal, por outro lado, pode ser aplicada ao raciocínio sobre o conhecimento e ação, como uma conseqüência de sua consideração com "necessidade" e "possibilidade". Essas considerações levaram os filósofos a introduzir a noção de um "estado consistente de associações" ou "universo possível". A necessidade corresponde à verdade em todos os universos possíveis, enquanto que a possibilidade corresponde à verdade em alguns universos possíveis. Pode-se olhar para o conhecimento e ação como associação de universos possíveis. Ou seja, os universos **u1** e **u2** podem estar relacionados pela consistência de **u2** com o que algum agente conhece em **u1** ou com o resultado de alguma ação realizada em **u1**.

Um problema mais grave é tornar tal lógica tratável computacionalmente. Uma solução envolve a tradução (isto é, axiomatização) para a lógica de primeira ordem, da interpretação de "universos possíveis". Estes "universos possíveis" então tornam-se coisas sobre as quais pode-se raciocinar.

2.10.5. Lógica temporal para raciocínio sobre o futuro

O raciocínio sobre a mudança - o que pode ser, o que poderia ser e o que poderia ter sido - tem um papel importante no comportamento conversacional. Quando uma pessoa não sabe a resposta para a pergunta de alguém, raciocina-se sobre se deveria sabê-lo depois. Se a resposta deixar a outra pessoa insatisfeita, raciocina-se sobre se deveria ter uma resposta melhor mais tarde. Por exemplo:

A: Érica matriculou-se em Física?

B_1: Eu não sei. Quer que eu o informe quando descobrir?

B_2: Não. Quer que eu o informe se ela se matricular no próximo ano?

É claro que é importante não fazer uma pergunta como essa:

A: Campinas está a menos de 30 quilômetros de Bragança Paulista?

B: Não, mas quer que eu o informe quando estiver?

Nem a lógica de primeira ordem com sua semântica padrão e nem a lógica modal, são adequadas para raciocínio sobre mudança. A lógica modal não provê uma boa representação em seqüências de eventos. Num sistema é necessário que possa-se raciocinar a partir de eventos passados para o que possa ser verdade mais tarde, incluindo possivelmente o presente. Computacionalmente, já existe uma abordagem para tornar o tempo uma representação implícita da entrada (ver item 3.3.4 sobre Redes Recorrentes).

2.10.6. O que é importante sobre a representação do conhecimento?

Em ciência da computação, uma boa solução freqüentemente depende de uma boa representação. Para a maioria das aplicações em Inteligência Artificial, a escolha da representação é mais difícil, pois as possibilidades são substancialmente maiores e os critérios são menos claros. A escolha da representação torna-se crucial para os estados do raciocínio e conhecimento de agentes inteligentes que podem entender a linguagem natural, pois as primitivas de representação e o sistema para sua combinação efetivamente limitam o que tais sistemas podem perceber, conhecer ou entender.

Os problemas para a elaboração de programas de computador inteligentes, que usam conhecimento para realizar tarefas, são muitos. Alguns desses problemas são (Woods, 1983):

- como estruturar um sistema de representação que será capaz, a princípio, de fazer todas as distinções importantes;

- como manter-se reservado sobre os detalhes que não podem ser resolvidos;

- como reconhecer, eficientemente, que conhecimento é relevante para o sistema, numa situação particular;

- como adquirir conhecimento dinamicamente, durante o tempo de vida do sistema; e

- como assimilar partes do conhecimento, na ordem em que elas são encontradas, ao invés de uma ordem específica de apresentação.

2.10.7. O papel da lógica na representação do conhecimento

Segundo Kolata (1982), os teóricos ainda não chegaram a um consenso em como resolver o problema da Inteligência Artificial - como tornar verdadeiras as máquinas pensantes. Ao invés disso, existem dois pontos de vistas filosóficos oponentes (A e B).

O ponto de vista A acredita que o caminho para resolver o problema é projetar programas de computador que raciocinem de acordo com as linguagens formais da lógica matemática, não importando se esta é ou não a forma como as pessoas pensam. O ponto de vista B acredita que uma abordagem favorável é tentar fazer com que os computadores imitem a forma como a mente humana trabalha, o que segundo este ponto de vista, quase não tem nada a ver com a lógica matemática.

Ambos os lados do debate concordam que o objetivo central da pesquisa é que os computadores devem, de alguma forma, vir a "conhecer" e "entender" o que todo humano conhece sobre o universo e sobre os organismos, naturais ou artificiais, que o habitam. Este corpo de conhecimento - indefinido, sem dúvida, nos seus limites - tem o nome de "senso comum". O problema que enfrenta-se é como colocar tal conhecimento em um robô. Isto é, como projetar um robô com uma capacidade de raciocínio suficientemente poderosa e favorável, que quando provido com algum subcorpo deste conhecimento, este seja capaz de gerar satisfatoriamente o resto desse conhecimento para adaptar-se inteligentemente e explorar seu ambiente? Pode-se assumir que a maioria do conhecimento de senso comum é geral, como no conhecimento de que objetos caem, a menos que estejam com algum suporte, de que objetos físicos não desaparecem de repente e de que fica-se molhado quando toma-se chuva.

2.10.8. O papel de uma rede de conhecimento para uma máquina inteligente

Um problema fundamental na construção de um agente de computador inteligente é a análise da situação para determinar o que fazer. Por exemplo, muitos sistemas especialistas são organizados em volta de um conjunto de "regras de produção", um conjunto de regras de ações-padrão, que caracterizam o comportamento desejado do sistema. Tal sistema opera, determinando em todo passo, que regras são satisfeitas pelo estado corrente do sistema, agindo então sobre este estado, executando uma dessas regras.

Segundo Woods, a abordagem ao problema de determinar que regras aplicar, assume que partes de padrões de todas as regras são organizadas em uma taxonomia estruturada de todas as situações e objetos, sobre os quais o sistema tem algum conhecimento. Por **taxonomia** entende-se uma coleção de conceitos ligados por uma relação generalizada, de tal forma que os conceitos mais gerais que um dado conceito, são acessíveis a partir dele. Por uma taxonomia **estruturada** entende-se que as descrições dos conceitos têm uma estrutura interna disponível ao sistema de computador tal que, por exemplo, a colocação dos conceitos dentro da taxonomia pode ser computacionalmente determinada. Uma característica de tal taxonomia é que a informação pode ser armazenada em seu nível mais geral de aplicabilidade e acessada indiretamente por conceitos mais específicos que "adquirem" aquela informação.

2.10.9. A necessidade de uma organização taxonômica

Para regras independentes, em um sistema de regras de produção clássico, os conflitos de várias regras serem satisfeitas ao mesmo tempo são descobertos apenas quando uma situação conflitante ocorre. Numa estrutura de classificação taxonômica, entretanto, a absorção das condições de uma regra por uma outra pode ser descoberta quando a regra é assimilada na taxonomia, quando a pessoa que entra com a regra introduz a questão de como as duas regras podem interagir.

A assimilação de regras em uma estrutura de conhecimento taxonômico não apenas facilita a descoberta de interações na ocasião da entrada, mas também promove uma compactação na especificação das regras. Confiando no fato de que conceitos adquirem informação a partir de

conceitos mais gerais, pode-se, usualmente, criar o conceito para a parte padrão de uma regra nova, apenas adicionando uma restrição pequena a um conceito existente.

2.10.10. Programação lógica como uma representação do conhecimento

A idéia de que a lógica serviria como uma linguagem de programação foi posta em prática em 1972, na forma do Prolog. Foi provado seu valor em várias áreas da computação, entre as quais o processamento de linguagem natural.

A introdução do Prolog tornou possível representar o conhecimento em termos da lógica e também fazer inferências a partir desse conhecimento, automaticamente.

2.11. A abordagem conexionista

2.11.1. O neurônio biológico

O neurônio "clássico" (figura 2.6A) tem muitos dendritos, usualmente ramificados, que recebem informação de outros neurônios e um único axônio que fornece como saída a informação processada, usualmente através da propagação de um "spike" ou "potencial de ação"[2]. O axônio divide-se em vários ramos que fazem sinapses[3] com os dendritos e corpos celulares de outros neurônios.

2.11.1.1. Variantes do neurônio "clássico"

Este quadro simples se torna complicado nas seguintes situações:

[2] O potencial de ação é um impulso numa fibra nervosa, que se move rapidamente ao longo do nervo.
[3] As junções entre as células nervosas são chamadas de sinapses. São os locais onde as células transferem sinais.

- um neurônio pode não ter axônios, mas apenas "processos" que servem tanto para receber como para transmitir informação (figura 2.6B).

- axônios podem formar sinapses em outros axônios (figura 2.6C);

- dendritos podem formar sinapses em outros dendritos (figura 2.6D);

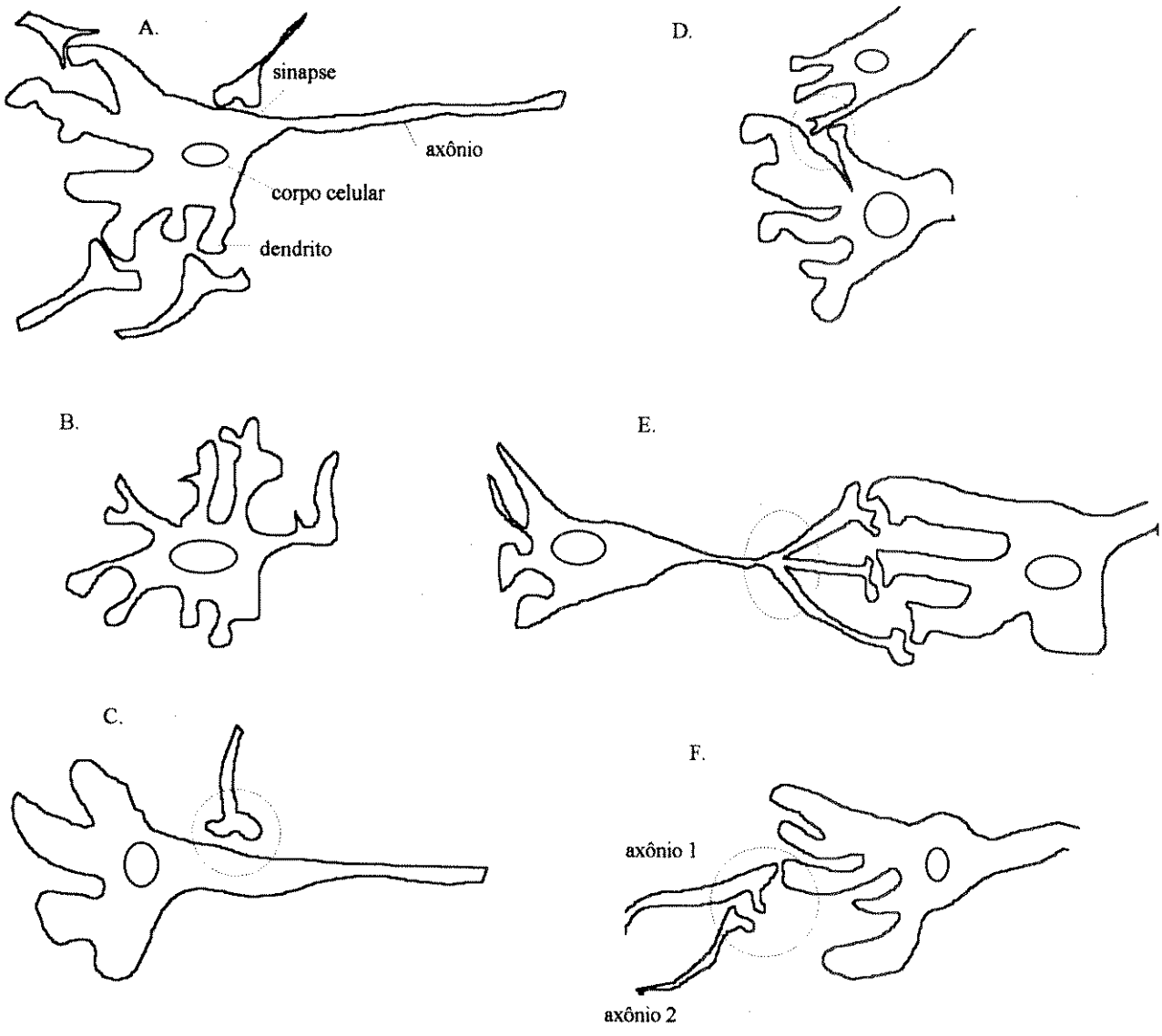


Figura 2.6. Diagramas esquematizados do neurônio clássico (A) e algumas de suas variantes (B - F) (McClelland e Rumelhart, 1986).

- um axônio pode não propagar um "spike" mas produzir um potencial de gradiente ("graded potential"). Por causa da atenuação, deve-se esperar que esta forma de sinalização da informação não ocorra através de distâncias longas (figura 2.6E). Estes potenciais de gradiente podem ocorrer em outro nível. Por exemplo, um terminal de axônio formando uma sinapse em uma dada célula pode receber uma sinapse (figura 2.6F). A sinapse pré-sináptica pode exercer apenas uma mudança de potencial local que é portanto restrito àquele terminal de axônio.

2.11.1.2. Sinapses: junções entre células nervosas

O tipo predominante de sinapse no cérebro do mamífero é a sinapse química, que opera através de liberação de uma substância transmissora do terminal pré-sináptico para o terminal pós-sináptico (figura 2.7).

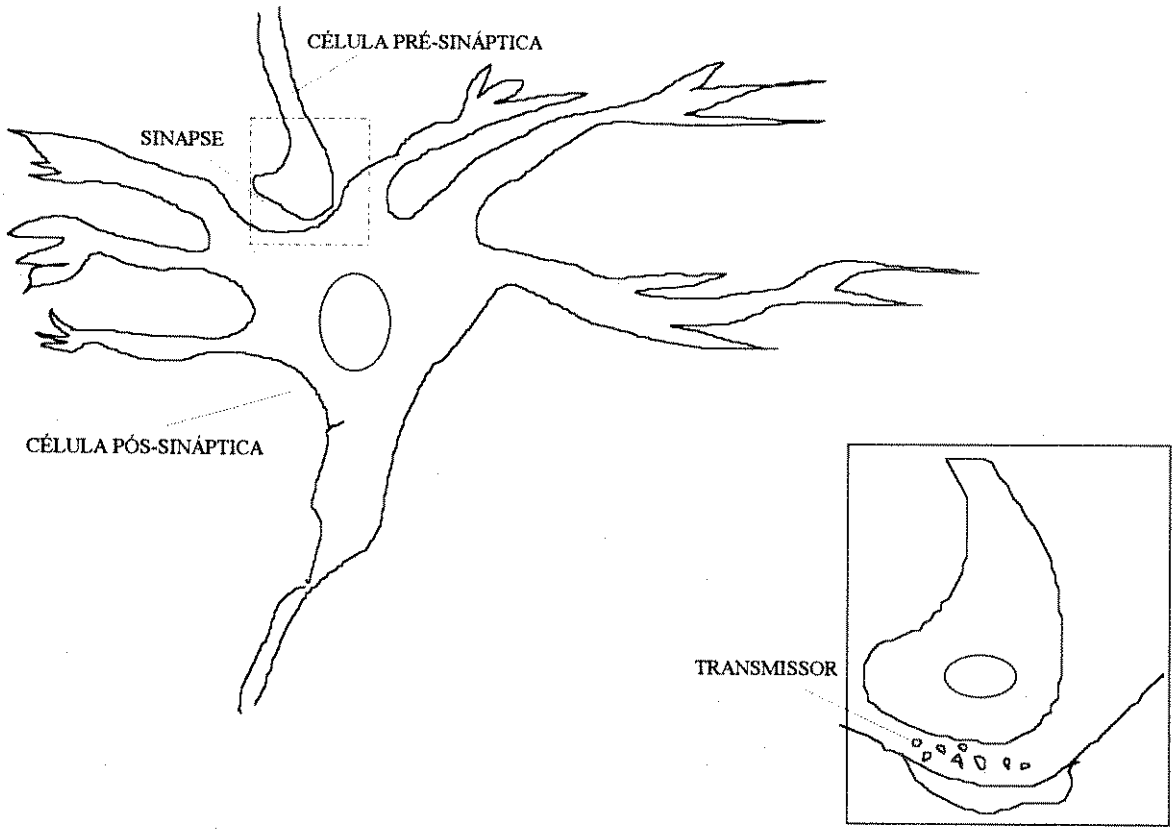


Figura 2.7. Na maior parte das sinapses, o terminal pré-sináptico libera uma substância química, o transmissor, em resposta a uma despolarização (Kuffler e Nicholls, 1976).

A acetilcolina (um neurotransmissor) é difundida por uma distância curta até a membrana pós-sináptica e age nas moléculas receptoras de acetilcolina específicas naquela membrana. Então, a acetilcolina é enzimaticamente dividida, onde parte dela é levada novamente à síntese de um novo transmissor.

As vesículas usadas fundem-se com a membrana do terminal pré-sináptico e novas vesículas são formadas da membrana nas margens do terminal.

2.11.1.3. As sinapses são químicas e não elétricas

Como já foi mencionado, a maior parte das sinapses que ocorrem no córtex cerebral são químicas e não elétricas. Os contatos sinápticos podem ser classificados morfologicamente em dois tipos básicos:

- tipo I (figura 2.8A): estas sinapses têm especializações de membrana assimétricas (a espessura da membrana é maior no lado pós-sináptico) e o processo pré-sináptico contém vesículas sinápticas redondas bastante grandes (50 nm), onde acredita-se que existam pacotes de neurotransmissores.

- tipo II (figura 2.8B): estas têm especializações de membrana simétricas. As vesículas sinápticas são menores e com os fixativos usuais usados pela microscopia eletrônica, são freqüentemente elipsoidais ou achatados. (A forma das vesículas depende dos detalhes de fixação e não é sempre um critério completamente confiável quando compara-se resultados relatados por diferentes pessoas.) A zona de contato é usualmente menor que da sinapse tipo I.

2.11.1.4. As sinapses podem excitar ou inibir

A importância da classificação nos dois tipos morfológicos é que as sinapses do tipo I parecem ser excitatórias, ao passo que as sinapses do tipo II parecem ser inibitórias[4].

[4] As células nervosas influenciam outras por (a) excitação, ou seja, elas produzem impulsos em outras células e (b) inibição, ou seja, elas previnem a liberação de impulsos em outras células.

Existe um outro critério possível para determinar o caráter das sinapses: o transmissor que elas usam. Em geral, assume-se que um dado transmissor fará usualmente a mesma coisa em lugares diferentes, apesar de haver exceções, dependendo da natureza dos receptores pós-sinápticos.

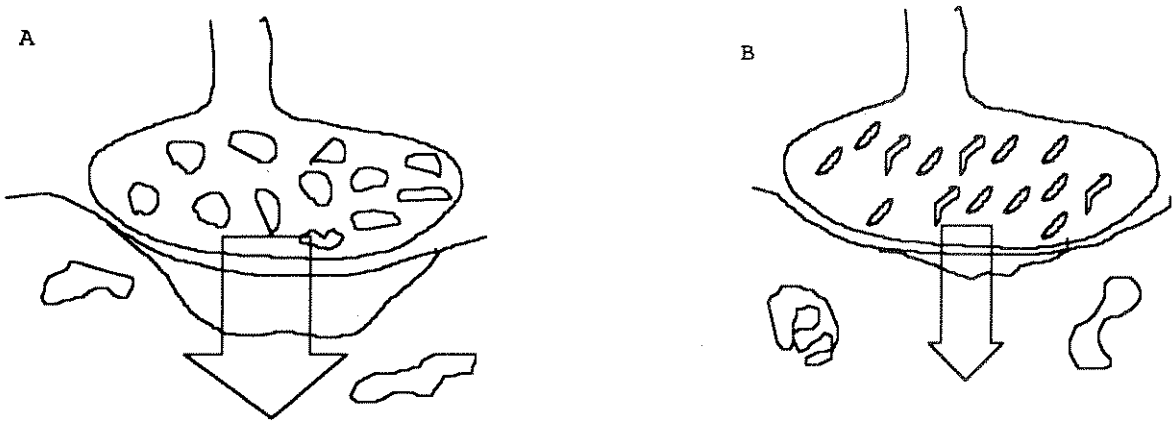


Figura 2.8. Diagramas idealizados das sinapses tipo I (A) e tipo II (B). Veja o texto para esclarecimentos (McClelland e Rumelhart, 1986).

2.11.1.5. Generalizações sobre Sinapses

Vários métodos têm sido usados para identificar os neurotransmissores, mas cada técnica tem limitações. No momento, é difícil identificar os transmissores envolvidos e seus efeitos pós-sinápticos em muitas sinapses do sistema nervoso central. Pode-se fazer uma lista de tentativas de possíveis generalizações sobre sinapses:

- nenhum axônio faz sinapses tipo I em alguns locais enquanto faz tipo II em outros;

- nenhum axônio no cérebro de mamífero mostrou liberação de dois neurotransmissores diferentes não peptídeos. (Mas parece que muitos neurônios, incluindo neurônios corticais, podem

liberar um transmissor "convencional" e um neuropeptídeo, ou em alguns casos, dois ou mais neuropeptídeos.);

- não existe evidência no cérebro de mamífero que um mesmo axônio possa causar excitação e inibição em sinapses diferentes, mas isto é certamente possível já que o efeito de um dado transmissor depende dos tipos dos receptores presentes e de seus canais de íon associados.

2.11.1.6. Peptídeos: moduladores da função sináptica

Ao longo dos últimos dez anos tem-se descoberto que existem muitos peptídeos distintos, de vários tipos e tamanhos, que podem agir como neurotransmissores. Há, no entanto, razões para suspeitar que os peptídeos são diferentes de muitos transmissores convencionais:

- peptídeos aparecem para "modular" a função sináptica ao invés de ativá-la;

- a ação de peptídeos, em poucos casos estudados, geralmente consiste em avançar vagarosamente e persistir por algum tempo, isto é, por segundos ou mesmo minutos, ao passo que os transmissores convencionais duram poucos milisegundos;

- em alguns casos foi mostrado que os peptídeos não agem onde foram liberados, mas a alguma distância. A difusão leva tempo. O tempo demorado de persistência seria compatível com os possíveis atrasos de tempo produzidos pela difusão;

- existem muitos exemplos agora conhecidos de um neurônio único produzindo, e presumivelmente liberando, mais de um neuropeptídeo.

2.11.1.7. Peptídeo: transmissor lento ou neuromodulador ?

Foi mostrado que os peptídeos formam um segundo, mais lento, meio de comunicação entre neurônios, mais econômico do que usar neurônios extras para este propósito.

Os peptídeos têm papel de modulação principalmente em sistemas neurais, nos quais o modo de comunicação é o endereçamento químico.

Como transmissores os peptídeos agem em locais bem restritos, mesmo assim como um meio de condução muito lento, não sustentando as altas freqüências dos impulsos. Como neuromoduladores da função sináptica, a sua atividade é mais intensa. Os efeitos excitatórios da substância P (um peptídeo) são muito lentos no início e prolongados na duração (mais de um minuto) e por si só não podem causar a despolarização[5] suficiente para excitar as células. O efeito, entretanto, é tornar os neurônios mais prontamente excitáveis por outras entradas excitatórias - um claro exemplo de "neuromodulação".

2.11.2. O cérebro como modelo

A idéia de simular o cérebro era o objetivo principal para muitos trabalhos iniciais em Inteligência Artificial. O cérebro era visto como uma "rede neural", ou seja, um conjunto de nós, ou neurônios, conectados por linhas de comunicação. Atualmente tem havido um crescente interesse no uso de modelos de redes neurais, ou conexionistas, como o campo é chamado. Modelos conexionistas são aplicáveis a vários problemas de ciência cognitiva, incluindo processamento de linguagem natural, processamento de fala e visão.

Um grande defensor do conexionismo é Daniel W. Hillis (1985). Sua Máquina de Conexão é mais parecida com o cérebro do que com um computador tradicional. Hillis argumenta que num computador convencional, a maior parte do silício fica inativo a maior parte do tempo. Num instante qualquer, apenas a CPU e poucos bytes de memória estão ativos. A máquina de conexão é composta de muitas unidades processador/memória, a maioria delas ativas ao mesmo tempo.

---

[5] Despolarização é uma redução do potencial da membrana celular para zero mV, sendo que o interior do neurônio torna-se mais positivo. A despolarização para um nível de potencial crítico, o limiar, causa o início de um impulso. No seu pico, o interior da célula torna-se positivo em relação ao seu exterior. Na maioria das sinapses, o terminal pré-sináptico libera uma substância química, o transmissor, em resposta a uma despolarização. Numa sinapse excitatória, o transmissor liberado pelo terminal pré-sináptico despolariza a célula pós-sináptica, fazendo com que o potencial de sua membrana atinja o limiar. Numa sinapse inibitória, o transmissor tende a manter o potencial da membrana da célula pós-sináptica abaixo do limiar.

Num nível mais simples, o cérebro funciona da seguinte forma: neurônios ativam ou inibem o disparo de outros neurônios. Se um determinado neurônio dispara ou não, depende das entradas inibitórias ou excitatórias de todos os neurônios conectados a ele. De alguma forma, as ativações de todos os neurônios que se comunicam entre si e a interação do sistema nervoso com o ambiente formam as lembranças e o pensamento.

2.11.2.1. O cérebro é relevante?

Uma linha de pesquisadores respeitáveis em Inteligência Artificial sente que estudar o cérebro não é a melhor maneira para se entender o pensamento. O cérebro representa apenas uma forma de se fazer uma máquina pensante. A Inteligência Artificial tradicional vê o pensamento como uma série de problemas para resolver e acredita piamente que não há razão filosófica para um computador não poder resolvê-los. A base para esta crença é a tese de Church-Turing, que estabelece que se uma função é computável, pode-se computá-la com um computador convencional - formalmente, uma máquina de Turing. Esta tese não pôde ser provada, mas é largamente aceita porque ninguém pôde pensar em um contra-exemplo.

2.11.2.2. Paralelismo

Uma outra razão para se estudar modelos parecidos com o cérebro é seu paralelismo. Os "circuitos" do cérebro são mais lentos do que os de um computador. Para que o cérebro trabalhe o mais rápido possível - os psicólogos mostraram que pode-se reconhecer objetos num segundo - muitos neurônios devem trabalhar em paralelo. Em contraste, muitos programas de Inteligência Artificial são muito lentos. Se se puder achar formas de se executar programas de Inteligência Artificial em paralelo, eles certamente seriam muito mais rápidos.

A computação paralela tem sido bastante explorada em ciência da computação nos últimos dez anos. As redes neurais representam apenas uma linha de pesquisa em computação paralela. Basicamente, deve-se responder duas questões fundamentais no projeto de um sistema de computador paralelo: como conectar os processadores para propósito de comunicação e quanto de potência computacional e memória cada processador deve ter.

Os pesquisadores de redes neurais pensam que seus modelos, por serem os mais fiéis sobre o cérebro conhecido, terão sucesso. Infelizmente, as redes neurais raramente têm sido construídas

em hardware; normalmente elas são simuladas por software. Estas simulações geralmente são muito lentas, pois um processador tem que fazer o trabalho de muitos. Até que construa-se hardware de processamento paralelo efetivo, os modelos conexionistas não alcançarão soluções eficientes para problemas da Inteligência Artificial.

2.11.2.3. Variedades de redes neurais

O percéptron foi um dos primeiros modelos de redes neurais. Um percéptron modela um neurônio tomando uma soma ponderada de suas entradas e enviando a saída 1 se esta soma for maior que um determinado limiar (senão, envia 0). Veja a figura 2.9.

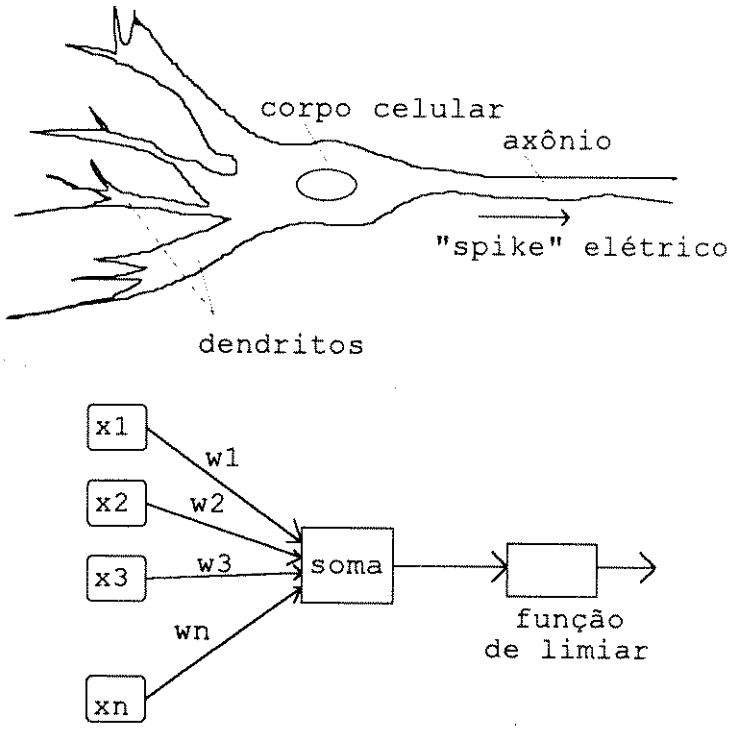


Figura 2.9. Um neurônio e um percéptron (Rich e Knight, 1991).

Muitos modelos de redes neurais devem alguma coisa aos percéptrons, apesar de serem mais gerais. O modelo típico de rede neural consiste de um conjunto de nós, ou neurônios, e conexões. Cada nó tem um número real associado, que é o seu limiar de ativação. Cada conexão contém também um número real, seu peso sináptico. Estes números são usualmente positivos e usualmente têm um valor máximo. Algumas unidades são conectadas à entrada e saída. Os pesos representam a força de conexão entre dois neurônios.

Geralmente, a rede neural é um sistema dinâmico que move-se de um estado para o próximo. Como tal, possui uma regra matemática que rege esse movimento. Um número infinito de tais regras é possível. Entretanto, normalmente quer-se limitar o modelo a influenciar a ativação de um dado nó baseado apenas nas ativações dos nós conectados a ele e nos pesos das conexões a esses nós.

As redes neurais não são explicitamente programadas como um computador convencional. Por melhor dizer, elas obedecem leis, ou regras, como um sistema físico. Deve-se programar um computador convencional mas uma rede neural simplesmente se conduz. Os projetistas de redes neurais vêem isto como uma vantagem, pois isto provê um mecanismo por meio do qual a inteligência pode surgir da lei física.

Uma das mais simples dessas regras é a regra linear. Computa-se a ativação de um dado nó como a soma dos produtos dos pesos de cada nó ao qual está conectado e a força dessa conexão. Essa regra é freqüentemente limitada: valores que passam de um certo limiar são cortados, para evitar valores de ativação grandes. Existem muitas variantes da regra linear.

Uma outra regra reforça a conexão entre dois nós que são altamente ativados ao mesmo tempo. Algumas versões desta regra de aprendizado permitem entradas, que ensinam, para influenciar a mudança de peso. Este tipo de regra é uma formalização da psicologia associacionista, que assegura que associações são acumuladas entre coisas que ocorrem juntas.

2.11.2.4. Aprendizado competitivo

O aprendizado é, talvez, o fenômeno mais importante em psicologia. Os primeiros pesquisadores em redes neurais eram ansiosos para mostrar como as redes podiam aprender padrões de entrada apresentados a elas - ou seja, como elas podiam vir a perceber esses padrões por elas mesmas.

Um dos métodos mais pesquisados nos últimos anos é o aprendizado competitivo. Este método tem um primeiro nível de unidades de entrada que contém o padrão a ser entrado no sistema. O nível acima das unidades de entrada consiste de "clusters"[6] de unidades. Cada unidade num "cluster" compete com as outras unidades no "cluster" pelo direito de reconhecer um padrão de entrada. Depois de um periodo de aprendizado, cada unidade num "cluster" reconhece um subconjunto dos padrões apresentados a ela. Portanto, cada "cluster" representa uma classificação, ou grupo, de padrões de entrada.

No aprendizado competitivo, cada unidade em cada "cluster" é conectada a todas as unidades de entrada. Os pesos das conexões são inicialmente colocados em valores aleatórios. Os pesos aleatórios fazem com que certas unidades nos "clusters" comecem a responder mais a determinados padrões de entrada, pois os pesos das conexões a essas unidades de entrada são mais fortes para alguns do que para outros.

No decorrer do aprendizado os pesos mudam. Como determinadas unidades no "cluster" tornam-se sensíveis a determinadas unidades no padrão de entrada, os pesos que conectam os pares associados de unidades aumenta, a custa de pares não associados. Unidades diferentes no mesmo "cluster" se inibem, de tal forma que apenas uma unidade num "cluster" ganha o direito de reconhecer um dado padrão.

Assim, com o tempo, unidades diferentes num "cluster" reconhecem propriedades diferentes de padrões de entrada. Por exemplo, um "cluster" de duas unidades pode separar todos os padrões de entrada naqueles que têm a maioria das suas unidades altamente ativadas e naqueles que têm a maioria desligadas. Os "clusters" maiores fariam mais classificações discriminatórias.

2.11.2.5. Máquinas de Boltzmann

Uma importante classe de redes neurais simula o comportamento de sistemas fisicos. Os sistemas fisicos têm uma tendência de moverem-se para dentro de estados de energia potencial mínima. Um exemplo simples disto é uma bola rolando num vale entre duas colinas. No alto da colina, a energia potencial é alta; no vale, é baixa.

---

[6] Conjuntos, ou grupos.

Este processo é chamado de relaxação. Mostrou-se que uma certa regra evolucionária simples para uma rede neural levará à relaxação. Sistemas como estes, que remontam aos sistemas termodinâmicos, como os átomos em uma sala, são chamados de máquinas de Boltzmann. As máquinas de Boltzmann são muito usadas em várias aplicações de redes neurais.

2.11.2.6. Representações distribuídas

Uma importante característica de muitos modelos de redes neurais é sua natureza distribuída. Uma rede semântica padrão, como aquelas usadas nos primeiros esquemas de representação de conhecimento, consiste de um conjunto de nós conectados de alguma forma. Cada nó representa uma única palavra ou conceito. Se a rede estiver "pensando" na palavra *gato*, o nó para *gato* é ativado e todos os outros nós não. Esta é uma representação local.

Em contraste, numa rede distribuída, os nós não têm um único significado; ou seja, um conceito individual é representado por um padrão por todos os nós. Por exemplo (Zeidenberg, 1987), se há dez nós, ativando os nós 1, 3, 4 e 7 pode-se representar o conceito *gorila*, enquanto que ativando os nós 1, 4, 5 e 7 pode-se representar o conceito próximo *chimpanzé*. Conceitos que são próximos têm representações similares.

Uma rede de processamento paralelo distribuído, uma rede neural que usa representação distribuída, oferece a vantagem de generalização automática. Se se quer representar o conceito *gorilas são cabeludos*, reforça-se a conexão entre todos os nós que compõem o conceito *gorila* e todos os nós que compõem o conceito *cabeludo*. Como resultado, desde que a maioria dos nós em *gorila* são também usados em *chimpanzé*, uma associação é também feita entre *chimpanzé* e *cabeludo*. É assim que a generalização automática trabalha. Numa representação local, onde *gorila* e *chimpanzé* são representados por nós separados, uma conexão entre *gorila* e *cabeludo* não implicaria numa conexão entre *chimpanzé* e *cabeludo*.

Uma outra vantagem de uma representação distribuída é sua insensibilidade a danos. Numa representação local, se o sistema perde o nó que representa *avó*, ele perde seu conceito de *avó*. Em uma representação distribuída, para perder um conceito, deve-se perder todos os nós que o representam. Se perde-se apenas um ou dois nós, o conceito pode degradar-se, mas ainda está lá. Isto é mais próximo ao tipo de memória perdida em adultos mais velhos.

2.11.2.7. Esquemas

Uma crítica aos modelos de redes neurais é que eles não são tão flexíveis na representação de conhecimento como os métodos padrões são. Os métodos padrões incluem a rede semântica local.

Os psicólogos cognitivos usam o conceito de esquema ("schema"). Um esquema é um espelho - na mente - de uma situação real. Como crianças, e como adultos, as pessoas aprendem novas associações e relações entre objetos, integrando-os ao esquema.

Não é imediatamente claro como um modelo de rede neural pode considerar o conhecimento representado em um esquema; entretanto, Rumelhart, Paul Smolensky, McClelland e Geoffrey Hinton (McClelland e Rumelhart, 1986) mostraram que é possível.

2.11.2.8. Hierarquias cognitivas

Freqüentemente, modelos de redes neurais são ordenados em hierarquias. Muitos níveis existem numa hierarquia, cada um composto de um conjunto de unidades. Tipicamente, as unidades que recebem a entrada estão no fundo do sistema (primeiro nível), e as unidades que fornecem a saída estão no alto (último nível). Num sistema "bottom-up", as unidades em cada nível são conectadas a outras unidades no seu próprio nível e influenciam as unidades nos níveis acima deles. Num sistema "top-down", as unidades novamente conectam-se a unidades de seu próprio nível, mas influenciam as unidades nos níveis abaixo.

"Top-down" e "bottom-up" são conceitos familiares em ciência cognitiva. Por exemplo, na percepção da sentença, estes termos referem a como elementos lingüísticos de tamanhos diferentes, o fonema (som), morfema (elemento palavra), palavra, sintagma e sentença, interagem um com o outro.

2.11.2.9. Uma rede de leitura paralela

Um problema na criação de uma rede de leitura é que as pessoas tendem a ler mais de uma palavra de cada vez. Uma rede simples não funciona, pois ela lê apenas uma palavra por vez. Como solução, McClelland e Rumelhart propõem cópias duplicadas de redes. Redes de

reconhecimento de palavras individuais duplicadas teriam conexões programáveis ao invés de conexões fixas por hardware ("hardwired") entre letras e palavras.

2.11.2.10. Processamento de sentenças

Um importante aspecto do entendimento de sentença envolve determinar os vários casos (papéis) que as partes diferentes de uma sentença têm. Por exemplo, considere as seguintes sentenças:

O macaco morreu.
O macaco quebrou.

Na primeira sentença, *macaco* é um animal, pois *morrer* é uma característica dos seres vivos; na segunda sentença, *macaco* é uma ferramenta para trocar pneus, pois um animal não *quebra*. De alguma forma o modelo deve discernir seus casos diferentes.

McClelland e Alan Kawamoto (1986) desenvolveram um sistema conexionista para fazer esta atribuição de casos. Palavras são descritas por **microcaracterísticas semânticas** - dimensões básicas que descrevem muitos objetos e ações. Por exemplo, duas das microcaracterísticas que descrevem substantivos são "humano" e "leveza", que têm os valores *humano*, *não-humano*, e *leve*, *pesado*, respectivamente. As palavras não são representadas diretamente nas redes do sistema mas em termos das ativações de unidades representando microcaracterísticas.

O modelo tem um grupo de unidades para cada um dos casos principais, que substantivos diferentes podem ter em uma ação. Estes casos são Agente (ator), Paciente (agido sobre), Instrumento (coisa usada) e Modificador (palavra adverbial ou cláusula). Por exemplo, a sentença *O homem comeu o sanduíche*, ativaria as microcaracterísticas de *comeu* e *homem* no conjunto das unidades que correspondem ao Agente; isto representa o fato de que o Agente para o verbo *comeu* é *homem*.

O sistema é treinado em uma série de sentenças. As atribuições de caso corretas para as sentenças de treinamento são mostradas ao sistema. Estas atribuições correspondem às ativações de nós particulares. O sistema ajusta as conexões entre esses nós de tal forma que eles reforcem-se mutuamente.

Depois de ser treinado com um número suficiente de sentenças, o sistema pode fazer atribuições de casos corretos para novas sentenças. Ele ainda pode fazer atribuições de casos corretos para sentenças com alguma ambigüidade sintática. Por exemplo, na sentença *O homem abateu o garoto com a maleta*, o sistema considera que *maleta* é o Instrumento de *abateu* ao invés de pertencer ao *garoto*, desde que *maleta* tenha microcaracterísticas que indiquem que ela é um instrumento.

O sistema proposto trabalha com esta abordagem e geralmente faz um bom trabalho em atribuição de casos.

2.11.2.11. O futuro

As redes neurais são boas para várias tarefas de processamento de linguagem natural, incluindo reconhecimento de letras, leitura e entendimento de sentenças. Elas também são úteis no armazenamento de conhecimento em esquemas e em recuperar itens da memória. A abordagem conexionista pode apontar uma solução para a Inteligência Artificial e psicologia cognitiva, forte e biologicamente plausível, para muitos problemas importantes.

Teoricamente, pode-se construir um modelo conexionista a partir do processo de entendimento de linguagem natural, desde que, como os psicólogos têm mostrado, envolva conhecimento integrado de muitos domínios, incluindo fonética, morfologia, sintaxe e semântica. Modelos conexionistas são particularmente bons na integração desses tipos de conhecimento.

2.12. O problema da integração

A interpretação da linguagem natural requer a aplicação cooperativa de muitos sistemas de conhecimento, conhecimento específico da linguagem sobre o uso da palavra, a ordem da palavra e a estrutura da frase, e o conhecimento do "mundo real" sobre situações estereotípicas, eventos, casos, contextos, etc. Não se pode construir um processador de linguagem natural psicologicamente realista meramente juntando vários módulos de processamento de conhecimento específico serialmente ou hierarquicamente.

Os problemas que surgem para a integração dos vários sistemas de conhecimento para o processamento de linguagem natural são os seguintes:

Ambigüidade. A ambigüidade é o maior problema no processamento de linguagem natural. Existem basicamente duas abordagens para tratamento de sentenças ambíguas: o "backtracking", como usado nas redes de transição aumentadas (ATNs) de Woods (1970) e o "delay", usado no analisador "espere e veja".

Interpretação única. Um outro fenômeno interessante na interpretação da linguagem é que as pessoas podem considerar apenas uma interpretação de uma sentença ambígua de cada vez, mas podem facilmente "saltar" entre interpretações.

Erros de compreensão. Erros na compreensão, como as sentenças enganosas[7], têm sido explicados por princípios estruturais puros. Entretanto, existem explicações mais completas e naturais como os efeitos colaterais de processos fortemente interativos. Os efeitos enganosos podem ocorrer em todos os níveis do processamento da linguagem. Considere a seguinte sentença:

O astrônomo casou-se com a estrela. (Charniak, 1983)

Os leitores usualmente reportam esta sentença como "temporariamente anômala", isto é, uma "sentença semanticamente enganosa". Uma explicação plausível para a dupla interpretação cognitiva da sentença é que o poder primitivo de "astrônomo" no sentido errado de "estrela" é inicialmente mais forte que o poder lógico das restrições de seleção de "estrutura de caso"; no final, as restrições de seleção forçam a interpretação de "estrela" como uma pessoa.

Texto não gramatical. As pessoas são capazes de interpretar linguagem não gramatical se esta ocorrer naturalmente (devido à gramática pobre, estrangeiros, interferência de barulho externo, interrupções, auto-correções, etc.).

---

[7] "garden path sentences".

2.13. Integrando fontes de conhecimento

Todos estes fenômenos indicam a necessidade de uma teoria de processamento de linguagem que afirme, ao invés de uma simples passagem de resultados incompletos entre componentes processadores, uma forte interação entre estes componentes de tal forma que todas as decisões sejam interdependentes. Acredita-se que a maioria das teorias de processamento de linguagem existentes são falhas, porque estão limitadas a um conjunto de idéias computacionais que efetivamente não podem tratar com decisões interdependentes e por causa das peculiaridades da língua portuguesa e da história da pesquisa lingüística que levou a assunções da autonomia da sintaxe no entendimento da linguagem natural.

Estas observações, é claro, não são inteiramente novas. No início dos anos 70, argumentava-se que a semântica, e não a sintaxe, deveria ter o papel central nas teorias programadas de processamento de linguagem natural.

2.14. Marcadores de Passo: Uma Teoria de Influência Contextual

A maior parte dos sistemas de compreensão de linguagem em Inteligência Artificial segue o modelo geral mostrado na figura 2.10 (Charniak, 1983). Um componente inicial usa conhecimento sintático para pegar a estrutura funcional da sentença (por exemplo, em "*Frederico foi morto por José*", necessita-se distingüir o matador do morto). O processo sintático é guiado por uma unidade semântica que também é responsável por tranformar a entrada em uma representação semântica. Para a maioria destes modelos, a informação passa entre sintaxe e semântica em alguma forma de árvore sintática.

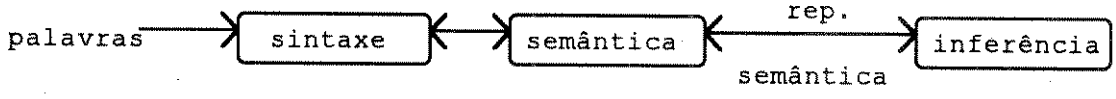


Figura 2.10. A Teoria Padrão (Charniak, 1983).

Vai-se tratar aqui dos seguintes casos:

(1) a determinação do "contexto" na compreensão da estória.
(2) o papel do contexto na não ambigüidade do sentido da palavra.
(3) A relação entre sintaxe e semântica.

Esta teoria está mostrada na figura 2.11.

Uma teoria de ativação alastradora (ou marcadores de passo) do processamento semântico humano, proposta inicialmente por Quillian (1968), além de ser uma teoria para explicar dados, era uma teoria projetada para mostrar como construir uma estrutura e processamento semânticos humanos em um computador.

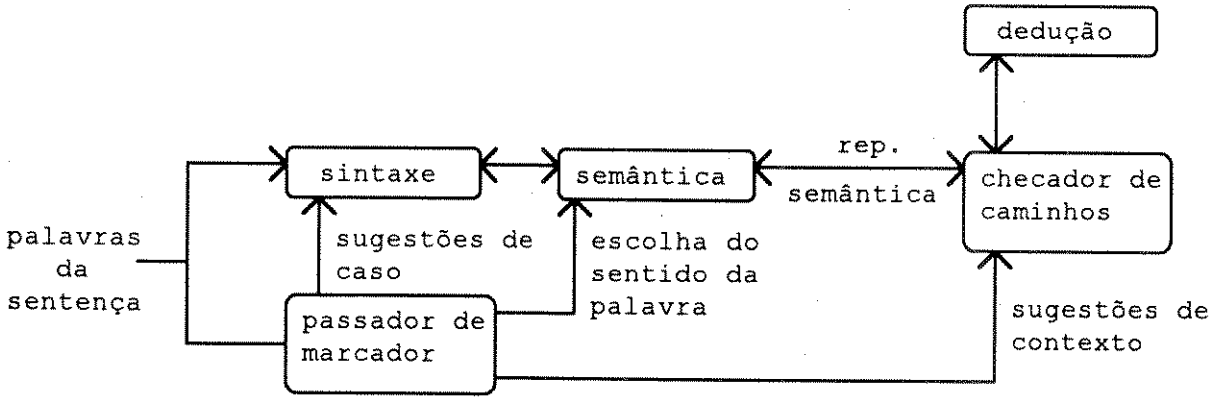


Figura 2.11. A Teoria proposta (Charniak, 1983).

2.14.1. A Teoria de Quillian da Memória Semântica

O fato de a teoria de Quillian ser desenvolvida para um computador digital impôs certas restrições à mesma.

Os conceitos das pessoas contêm quantidades indefinidamente grandes de informação. Quillian usava o exemplo de uma máquina. Se alguém pedir para uma pessoa contar tudo que ela sabe sobre máquinas, ela vai começar dando propriedades óbvias, por exemplo, que as máquinas são feitas pelo homem e que têm partes móveis. Mas logo a pessoa vai desprezar os fatos óbvios e, cada vez mais, vai fornecer fatos que são menos relevantes, por exemplo, que uma furadeira é uma máquina, etc. A quantidade de informação que uma pessoa pode gerar sobre qualquer conceito desta forma parece ilimitado.

Um conceito pode ser representado como um nó numa rede, com as propriedades do conceito representadas como ligações relacionais rotuladas a partir do nó para outros nós de conceito. Estas ligações são ponteiros e usualmente vão em ambas as direções entre dois conceitos. As ligações podem ter **critérios** diferentes, que são números indicando o quão essencial é cada ligação para o significado do conceito. Os critérios em qualquer par de ligações entre dois conceitos podem ser diferentes; por exemplo, deve ser de alto critério para o conceito de uma furadeira ser uma máquina, e de baixo critério para o conceito de um tipo de máquina ser uma furadeira. A partir de cada um dos nós ligados a um nó dado existirão ligações para outros nós de conceito e a partir de cada um destes nós para outros e assim por diante.

2.15. Determinação Contextual de Sentidos de Palavras

Na presença do contexto certo, as pessoas não estão conscientemente preparadas para as palavras ambíguas. Nunca se considera todas as alternativas. As coisas que parecem ser organizadas têm o significado certo considerado primeiro. Porém no exemplo de Charniak:

O astrônomo casou-se com a estrela

a maioria das pessoas tem dificuldade de achar a leitura de "estrela de cinema" para "estrela", a despeito do fato de que a leitura de "objeto astronômico" torna difícil imaginar a cerimônia de casamento. Este exemplo evidencia uma teoria na qual as pessoas consideram que os sentidos das palavras são modificados por itens contextuais (ou mesmo que apenas alguns sentidos estão "disponíveis" para consideração). De acordo com esta teoria, os problemas com exemplos deste

tipo vêm do fato que o primeiro sentido tentado não está de acordo com os requisitos reais da sentença e portanto deve ser rejeitado.

Através dos experimentos de Swinney (1979), conclui-se que o contexto não pré-seleciona quais sentidos de palavras considerar. Mas se este é realmente o caso, por que as pessoas têm problemas com o exemplo do "astrônomo/estrela"? Faz sentido quando assume-se que o contexto pré-selecionou o sentido incorreto da palavra. Mas foi dito que o contexto não pré-seleciona. Então parece que a semântica tem todos os sentidos disponíveis para fazê-lo e tem todas as informações necessárias para fazer a escolha certa (em particular, ela sabe que "estrela" é o objeto de "casar" e que "casar" requer uma pessoa como objeto). Ainda que a semântica faça a escolha errada. No modelo padrão da figura 2.10, não há razão para isto acontecer. Por que então as pessoas erram?

2.16. Gramáticas de Cláusulas Definidas: Introdução

Uma forma de se definir uma **linguagem**, com precisão, seja uma linguagem natural ou uma linguagem de programação, é através de uma coleção de **regras** chamada de **gramática**. As regras de uma gramática definem quais cadeias de palavras ou símbolos são **sentenças** válidas da linguagem. Além disso, a gramática geralmente fornece algum tipo de **análise** da sentença, em uma estrutura que torna seu significado mais explícito.

Uma classe fundamental de gramática é a **gramática livre de contexto** (GLC), ou "forma de Backus-Naur" (BNF). Nas GLCs, as palavras, ou símbolos básicos, da linguagem são identificados por símbolos **terminais**, enquanto que sintagmas da linguagem são identificados por símbolos **não-terminais**. Cada regra de uma GLC expressa uma forma possível para um não-terminal, como uma seqüência de terminais e não-terminais. A análise de uma cadeia de acordo com a GLC é uma **árvore de análise**, mostrando os sintagmas constituintes da cadeia e seus relacionamentos hierárquicos.

Uma idéia importante é traduzir o formalismo de propósito especial de GLCs em um formalismo de propósito geral, chamado de lógica de predicado de primeira ordem. Foi elaborado um método particular para expressar regras livres de contexto como comandos lógicos de um tipo

restrito, conhecido como **cláusulas definidas** ou "cláusulas de Horn". O problema de reconhecer, ou analisar, uma cadeia de uma linguagem é então transformado no problema de provar que um certo teorema segue dos axiomas de cláusulas definidas que descrevem a linguagem.

Estas idéias poderiam apenas ser de interesse teórico. Entretanto, ao mesmo tempo, originou-se uma idéia de maior alcance. Uma coleção de cláusulas definidas pode ser considerada um **programa**. Isto resulta que a dedução automática pode exibir todas as características associadas à computação efetiva, na condição de que a dedução é obtida numa forma direcionada à meta.

Uma realização prática deste conceito de "programação em lógica" foi desenvolvida na forma da linguagem de programação Prolog. Prolog é baseada num procedimento de prova bem simples, mas muito eficiente. Muitas implementações da linguagem existem e estas implementações têm mostrado que Prolog é tão eficiente quanto as linguagens de programação de alto nível convencionais.

Agora, se uma GLC é expressa em cláusulas definidas de acordo com este método e executada como um programa Prolog, o programa comporta-se como um analisador "top-down" eficiente, para a linguagem que a GLC descreve[8]. Este fato torna-se particularmente insignificante quando combinado com uma outra descoberta - que a técnica para traduzir as GLCs em cláusulas definidas tem uma generalização simples, resultando em um formalismo mais poderoso que as GLCs, mas igualmente pronto para execução pelo Prolog. Este formalismo chama-se **gramáticas de cláusulas definidas** (GCDs). As GCDs são um caso especial das "gramáticas de metamorfose", que estão para as gramáticas de Chomsky do tipo 0 assim como as GCDs estão para as GLCs. Ainda que as gramáticas de metamorfose possam ser traduzidas em cláusulas definidas, a correspondência não é tão direta quanto para as GCDs.

As GCDs são uma extensão natural das GLCs. Como tais, as GCDs herdaram as propriedades que fazem as GLCs tão importantes para a teoria da linguagem: as formas possíveis para as sentenças de uma linguagem são descritas numa forma clara e modular; é possível representar as características recursivas dos sintagmas, comuns a quase todas as linguagens de interesse; existe um corpo estabelecido de resultados nas GLCs que é muito útil no projeto de algorítmos de análise.

---

[8] A eficiência do analisador também depende de uma escolha apropriada da GLC para descrever a linguagem.

É sabido que as GLCs não são totalmente adequadas para descrever linguagem natural, nem mesmo muitas linguagens de programação. As GCDs compensam este problema estendendo as GLCs em três importantes formas (Pereira e Warren, 1980).

Primeiro, as GCDs provêem dependência de contexto numa gramática, tal que as formas permissíveis para um sintagma podem depender do contexto no qual este sintagma ocorre na cadeia. Segundo, as GCDs permitem que estruturas de árvore arbitrárias sejam construídas no decorrer da análise, numa forma que não é forçada pela estrutura recursiva da gramática; tais estruturas de árvore podem prover uma representação do "significado" da cadeia. Terceiro, as GCDs permitem condições extras a serem incluídas nas regras da gramática; estas condições fazem o curso da análise depender de cálculos auxiliares.

2.17. Entendimento de Linguagem Natural

As pessoas confundem "linguagem natural" com "linguagem humana" e os computadores simplesmente não estão adequados ainda para a interação humana. Para ser capaz de entender a linguagem humana, um computador necessitaria possuir o tipo de conhecimento sobre a linguagem que as pessoas possuem.

Uma gramática para linguagem humana assegura que as partes da sentença concordam em tempo verbal, pessoa, etc., isto é referido como informação de "contexto". As GLCs não são capazes de manipular a informação de contexto mas as GCDs sim. Portanto, quando construídas com GCDs, as linguagens naturais podem imitar a linguagem humana.

Além da complexidade, as linguagens humanas são grandes demais para os computadores trabalharem. Para uma conversação na linguagem humana, um computador teria que armazenar as definições de milhares de palavras e todas as formas possíveis nas quais estas palavras podem ser combinadas.

Ainda que o tamanho e a complexidade das linguagens humanas não possam ser manipuladas pela tecnologia corrente, sistemas de linguagem "naturais" úteis, porém limitados, podem ser desenvolvidos. Se o sistema é limitado de tal forma que ele trabalhe apenas com um comando de

cada vez, a gramática é menos complexa. Isto torna possível a construção de um sistema usando técnicas bem conhecidas, tais como as GCDs. Se os tópicos sobre os quais o sistema pode conversar são também limitados, a gramática é pequena.

Mesmo com essas limitações, ainda existem problemas. As linguagens humanas permitem anomalias que as linguagens naturais não podem permitir. Uma anomalia que as linguagens humanas permitem é a referência ambígua. Por exemplo, na sentença "O sorvete está no freezer e ele está gelado," deve-se decidir se "ele" refere ao sorvete ou ao freezer. Pode-se usualmente adivinhar ao que "ele", "isto" e "aquilo" referem, simplesmente pelo contexto. Estas regras de adivinhação não são facilmente traduzidas em regras gramaticais.

Ainda que palavras como "ele", "isto" e "aquilo" alertem para referências ambíguas, outras referências ambíguas são menos óbvias de se ver. Considere a sentença "Eu conheço uma mulher com uma perna de madeira chamada Dalva". Sabe-se que Dalva é o nome da mulher e não da perna de madeira. Entretanto, o sistema de linguagem natural precisaria saber que as pessoas têm nomes e as pernas de madeira normalmente não.

Na linguagem humana, as partes de uma sentença podem ser arranjadas de diferentes formas e ainda assim produzir o mesmo significado. Por exemplo, "A que horas o trem parte?" e "O trem parte a que horas?" significam a mesma coisa. Entretanto, cada forma requer sua própria definição de gramática.

Um sistema de linguagem natural que possa manipular todas essas situações seria grande e complexo demais. Os sistemas de linguagem natural não devem apenas limitar a gramática a sentenças simples, eles devem também fazer assunções sobre os tipos de sentenças que o usuário entrará. Quando se projeta um sistema de linguagem natural, deve-se decidir quais formas de sentenças o sistema reconhecerá e o que estas sentenças significarão. Freqüentemente, dentro de uma área bem definida, pode-se escolher um número limitado de estruturas de sentenças para atender a maioria das necessidades do usuário.

2.18. Ferramentas para escrever o analisador

Uma introdução breve e informal a programas lógicos e gramáticas de metamorfose é dado aqui.

2.18.1. Programação lógica

Programas lógicos são essencialmente conjuntos de cláusulas da forma:

$$B \leftarrow A1,A2,...,An$$

chamadas de cláusulas de Horn, onde *B* e *Ai* são fórmulas atômicas e todas as variáveis nas fórmulas atômicas são assumidas universalmente quantificadas. "←" é lido "se" e as vírgulas significam conjunção. Um lado direito vazio denota uma asserção não condicional do fato. Por exemplo,

1) gosta(mãe(*x*),*x*) ←
"todo *x* é gostado(a) por sua mãe"

2) gosta(roberto,*y*) ← gosta(*y*,roberto)
"Roberto gosta de todo *y* que gosta dele"

As variáveis aparecem em itálico para distingüir das constantes.

Com respeito a um dado conjunto de cláusulas (isto é, um programa), o usuário pode perguntar por relações a serem calculadas, colocando chamadas de procedimentos, isto é, cláusulas da forma:

$$\leftarrow A1,A2,...,An$$

Isto inicia um processo de demonstração automático, durante o qual as variáveis na chamada tomam valores para os quais "A1 e A2 e ... An" contêm. Aqui "←" pode ser interpretada como uma interrogação. Uma terminação mal sucedida implica que não existem tais valores.

Portanto, com respeito as cláusulas 1 e 2 acima, a seguinte chamada:

3) ← gosta(*z*,roberto)
"Quem gosta de Roberto?"

resulta em *z* sendo unificado (ligado) a "mãe(roberto)". O mesmo resultado teria sido obtido da chamada:

4) ← gosta(*z*,roberto),gosta(roberto,*z*)
"Quem gosta de e é gostado por Roberto?"

Resultados alternativos para a mesma chamada podem ser obtidos com programas não determinísticos. Por exemplo, se se adicionasse a cláusula:

gosta(sonia,roberto) ←

então chamar 3 pode alternativamente resultar em *z* sendo ligado a "sonia".

Interpretadores práticos de programas lógicos, tal como Prolog, também incluem algumas características extra-lógicas para funções de entrada e saída e de controle.

2.18.2. Gramáticas de metamorfose

As GMs são um formalismo poderoso para descrever e processar linguagens, nas quais:

- regras de reescrita sensíveis ao contexto podem ser descritas.

- qualquer substring de qualquer tamanho pode ser reescrita em qualquer outra string.

- símbolos de gramática podem incluir argumentos.

- condições na aplicação de regras podem ser especificadas.

- geração e análise de sentença são fornecidos pelo processador.

As GMs podem ser consideradas como uma notação alternativa conveniente para programas lógicos. Ao invés de definí-las precisamente, exibe-se algumas regras gramaticais e mostra-se informalmente uma forma de traduzí-las para as cláusulas de Horn, que basicamente segue a axiomatização das GMs do Prolog.

2.18.3. GMs normalizadas

Prolog apenas aceita regras GM **normalizadas**, isto é, da forma

$$A\ x \rightarrow y$$

onde A é um símbolo não terminal, x é uma cadeia de terminais e y é qualquer seqüência de símbolos gramaticais. Esta restrição é necessária, dentro do esquema mostrado, para traduzir regras em cláusulas de **Horn** (também chamadas de cláusulas definidas), nas quais no máximo uma fórmula atômica do lado esquerdo é permitido. Por esta razão, elas também são chamadas de gramáticas de cláusulas definidas (GCDs)[9].

As regras não normalizadas podem ser facilmente substituídas por um conjunto equivalente de regras normalizadas. Por exemplo, B a C b $\rightarrow$ f g pode ser substituída por B a c b $\rightarrow$ f g e C $\rightarrow$ c, onde c é um terminal "de imitação". Do ponto de vista da análise, os resultados são equivalentes.

Vai-se ignorar entretanto esta restrição no que segue, para tornar mais claro o texto.

[9] Num sentido restrito, as gramáticas de cláusulas definidas (GCD) permitem apenas um único não-terminal do lado esquerdo e portanto, uma forma ainda mais restrita de GMs.

2.18.4. Grafos de derivação

Ainda que as GMs possam ser entendidas declarativamente, algumas vezes é útil seguir uma derivação completa da sentença, construindo um grafo que descreva a história das aplicações de regra "top-down" e da esquerda para a direita. Isto é ilustrado através da gramática:

1) Sentença(s) → Nome-próprio(k),Verbo(k,s).
2) Nome-próprio(tom) → tom
3) Nome-próprio(joão) → joão
4) Verbo(k,rir(k)) → ri

O grafo de derivação para "João ri" é mostrado na figura 2.12. Os números identificam a regra aplicada. As substituições necessárias para aplicar as regras aparecem do lado direito dos rótulos. Através delas pode-se reconstruir a estrutura profunda "rir(joão)".

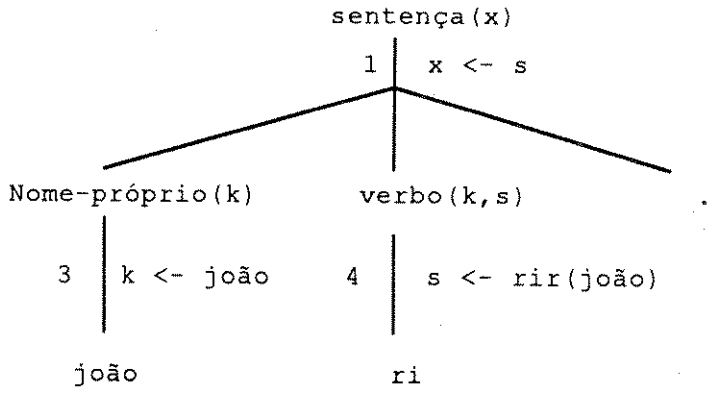


Figura 2.12. Grafo de derivação para "João ri" (Dahl, 1981).

Note que, quando a variável recebe um valor, este valor é propagado para cada uma das suas ocorrências no grafo. Então, quando se aplica a regra 4, sabe-se que o valor de k=joão. Também, a renomeação de variáveis deve acontecer sempre que necessária a fim de assegurar que a regra aplicada e a cadeia que ela aplica não compartilhem nenhuma variável.

2.19. Lógica como uma linguagem de programação - O subconjunto de cláusulas definidas

Vai-se definir aqui a sintaxe e a semântica de um certo subconjunto da lógica ("cláusulas definidas"), que é basicamente uma porção de disjunções a partir da lógica e indica-se como esse subconjunto forma a base da linguagem de programação prática conhecida como Prolog. As cláusulas definidas também têm sido chamadas de "cláusulas de Horn" ou "cláusulas regulares". Descrever-se-á o subconjunto de cláusulas definidas a partir do ponto de vista de uma linguagem de programação convencional, usando a notação e a terminologia do Prolog.

2.19.1. Sintaxe, terminologia e semântica informal

2.19.1.1. Termos

Os objetos de dados da linguagem são chamados de **termos**. Um termo ou é uma **constante**, uma **variável** ou um **termo composto**.

As constantes incluem os **inteiros** tais como:

0, 1 ou 999

e **átomos** tais como:

'atomo'

O símbolo para um átomo pode ser qualquer seqüência de caracteres, que em geral deve ser escrita entre apóstrofes a menos que não haja possibilidade de confusão com outros símbolos (tais como variáveis ou inteiros). Como nas linguagens de programação convencionais, as constantes denotam objetos elementares definidos.

As variáveis serão distingüidas por uma letra inicial maiúscula, por exemplo:

X, Valor, A, A1

Uma variável deve ser vista como algum objeto particular mas não identificado. Note que uma variável não é simplesmente uma posição de armazenamento à qual pode-se atribuir um valor, como na maioria das linguagens de programação; ao invés disso, ela é um nome local para algum objeto de dado.

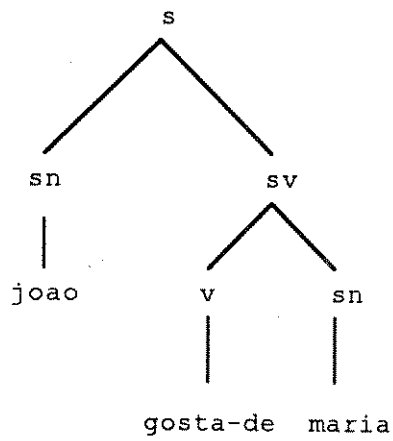
Os objetos de dados estruturados da linguagem são os termos compostos. Um termo composto compreende um **functor** (chamado o functor principal do termo) e uma seqüência de um ou mais termos chamados **argumentos**. Um functor é caracterizado por seu nome, que é um átomo, e sua **aridade** ou número de argumentos. Por exemplo, o termo composto cujo functor chama-se 'ponto' de aridade 3, com argumentos X, Y e Z, escreve-se:

ponto(X,Y,Z)

Pode-se imaginar que um functor seja um tipo registro e os argumentos de um termo composto sejam os campos do registro. Os termos compostos são usualmente representados graficamente como árvores. Por exemplo, o termo:

s(sn(joão),sv(v(gosta-de),sn(maria)))

seria representado como a estrutura:



Algumas vezes é conveniente escrever um termo composto usando uma notação infix opcional, por exemplo:

$$X + Y \ (P;Q) \ X < Y$$

ao invés de:

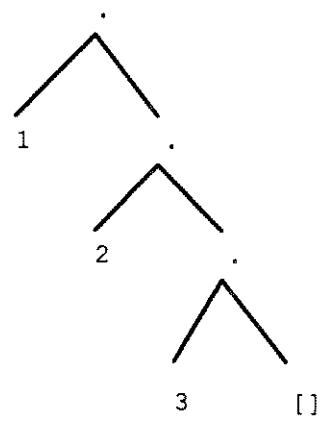
$$+(X,Y) \ ;(P,Q) \ <(X,Y)$$

Note que um átomo é considerado um functor de aridade 0.

Uma classe de estruturas de dados importante são as listas. Uma lista é o átomo

[]

representando a lista vazia, ou é um termo composto com o functor '.' e dois argumentos que são respectivamente a cabeça e a cauda da lista. Então a lista dos primeiros três números naturais é a estrutura:



Isto seria escrito na sintaxe padrão como

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

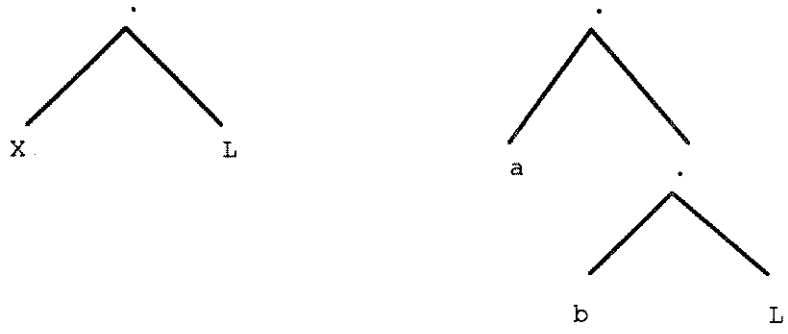
mas pode-se escrevê-lo usando uma notação de lista especial, como

[1,2,3]

A notação, quando a cauda da lista é uma variável, é exemplificada por

[X|L] [a,b|L]

representando respectivamente



2.19.1.2. Cláusulas

Uma unidade fundamental de um programa lógico é a **meta** ou **chamada de procedimento**. Exemplos são:

dá(tom,maçã,professor) reverso([1,2,3],l) X < Y

Uma meta é um tipo especial de termo, distingüido apenas pelo contexto no qual ele aparece no programa. O functor (principal) de uma meta é chamado de **predicado**. Ele corresponde a um nome de procedimento numa linguagem de programação convencional.

Um **programa** lógico consiste simplesmente de uma seqüência de comandos chamados de **cláusulas**. Uma cláusula compreende uma **cabeça** e um **corpo**. A cabeça ou consiste de uma meta simples ou é vazia. O corpo consiste de uma seqüência de zero ou mais metas (isto é, ele também pode ser vazio).

Se nem a cabeça e nem o corpo de cláusula são vazios, a cláusula é chamada de não-unitária (**regra**), e escrita na forma:

P :- Q,R,S.

onde P é a meta cabeça e Q, R e S são as metas que fazem parte do corpo. Pode-se ler tal cláusula ou **declarativamente** como:

"P é verdadeiro se Q e R e S são verdadeiros"

ou **procedimentalmente** como:

"Para satisfazer a meta P, satisfaça as metas Q, R e S."

Se o corpo da cláusula está vazio, a cláusula é chamada unitária (**fato**), e escrita na forma:

P.

onde P é a meta cabeça. Isto é interpretado declarativamente como:

"P é verdadeiro."

e procedimentalmente como:

"Meta P é satisfeita."

Finalmente, se a cabeça da cláusula é vazia, chama-se a cláusula de **questão** e escreve-se na forma:

?-P,Q.

onde P e Q são as metas do corpo. Tal questão é lida declarativamente como:

"P e Q são verdadeiros?"

e procedimentalmente como:

"Satisfaça as metas P e Q."

As cláusulas geralmente contêm variáveis. Note que as variáveis em cláusulas diferentes são completamente independentes, mesmo que elas tenham o mesmo nome - isto é, o "escopo léxico" de uma variável é limitado a uma única cláusula. Cada variável distinta em uma cláusula deveria ser interpretada como contendo um valor arbitrário. Para ilustrar isto, tem-se alguns exemplos de cláusulas contendo variáveis, com possíveis leituras declarativas e procedimentais, segundo Pereira e Warren:

(1) empregado(X) :- emprega(Y,X).

"Para quaisquer X e Y, X é empregado se Y emprega X."
"Para descobrir se X é empregado, ache um Y que emprega X."

(2) derivativo(X,X,1).

"Para qualquer X, o derivativo de X com respeito a X é 1."
"A meta de achar um derivativo para a expressão X com respeito a X é satisfeita pelo resultado 1."

(3) ?-tem_casco(X),aquático(X).

"É verdadeiro para qualquer X, que X tem casco e X é aquático?"
"Ache um X que tem casco e é aquático."

Num programa lógico, o **procedimento** para um predicado particular é a seqüência de cláusulas no programa cujas metas cabeça têm aquele predicado como functor principal. Por exemplo, o procedimento para o predicado ternário 'concatenar' deve consistir de duas cláusulas:

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

onde 'concatenar(L1,L2,L3)' significa "a lista L1 concatenada com a lista L2 é a lista L3".

Como se viu, as metas no corpo de uma cláusula são ligadas pelo operador ',' que pode ser interpretado como conjunção. Por conveniência, algumas vezes pode-se usar um operador ';' significando disjunção. (A precedência de ';' é tal que ele domina ',' mas é dominado por ':-'). Um exemplo é a cláusula:

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

que pode ser lido como:

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

Pode-se eliminar o uso de disjunção, definindo um predicado extra - ou seja, o exemplo anterior é equivalente a:

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

- e tal disjunção não será mencionada na descrição da semântica de cláusulas mais formal que se segue.

2.19.2. Semântica declarativa e procedimental

A semântica das cláusulas definidas deveria estar clara a partir das interpretações informais já dadas. Entretanto, é útil ter uma definição precisa. A **semântica declarativa** de cláusulas definidas revela quais metas podem ser consideradas verdadeiras de acordo com um dado programa e é definida recursivamente como se segue.

Uma meta é **verdadeira** se ela é a cabeça de alguma instância da cláusula e cada uma das metas (se existir) no corpo desta instância da cláusula forem verdadeiras, onde uma **instância** de uma cláusula (ou termo) é obtida, pela substituição, para cada zero ou mais variáveis da cláusula, de um novo termo para todas as ocorrências da variável.

Por exemplo, segundo Pereira e Warren, se um programa contém o procedimento precedente para 'concatenar', então a semântica declarativa diz que

concatenar([a],[b],[a,b])

é verdadeiro, porque sua meta é a cabeça de certa instância da primeira cláusula para 'concatenar', ou seja,

concatenar([a],[b],[a,b]) :- concatenar([],[b],[b])

e sabe-se que a única meta no corpo desta instância de cláusula é verdadeira, já que é uma instância do fato, que é a segunda cláusula para 'concatenar'.

Note que a semântica declarativa não faz referência à seqüencia de metas dentro do corpo de uma cláusula, nem à seqüencia de cláusulas dentro de um programa. Esta informação de seqüenciamento é, entretanto, muito relevante para a **semântica procedimental** que Prolog dá às cláusulas definidas. A semântica procedimental define exatamente como o sistema Prolog executará uma meta e a informação de seqüenciamento é a forma pela qual o programador Prolog direciona o sistema para executar seu programa. O efeito de executar uma meta é enumerar, uma por uma, suas instâncias verdadeiras. Aqui está uma definição informal da semântica procedimental.

> Para **executar** uma meta, o sistema procura a primeira cláusula cuja cabeça **casa** ou **unifica** com a meta. O processo de **unificação** acha a instância comum mais geral de dois termos, que é única se existir. Se um casamento é encontrado, a instância da cláusula casada é então **ativada** executando por sua vez, da esquerda para a direita, cada uma das metas (se existir) em seu corpo. Se, alguma vez, o sistema falha ao achar um casamento para uma meta, ele retorna ("**backtracks**"), isto é, ele rejeita a cláusula ativada mais recente, desfazendo quaisquer substituições feitas pelo casamento com a cabeça da cláusula. Depois ele reconsidera a meta original que ativou a cláusula rejeitada e tenta achar uma cláusula subseqüente que também casa com a meta.

Por exemplo, considere a questão:

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

que pode ser lida declarativamente como:

"Existem as listas X e Y que quando concatenadas fornecem a lista [a,b]?"

Se se executar a meta expressa nesta questão, acha-se que ela casa com a cabeça da primeira cláusula para 'concatenar', com X instanciado a [a | X1]. A nova variável X1 é restrita pela nova meta (ou chamada de procedimento recursivo) que é produzido:

concatenar(X1,Y,[b])

Novamente esta meta casa com a primeira cláusula, instanciando X1 a [b | X2], e fornecendo a nova meta:

concatenar(X2,Y,[])

Agora esta meta casará apenas com a segunda cláusula, instanciando X2 e Y a []. Já que não existem mais metas a serem executadas, tem-se a solução:

X = [a,b]
Y = []

isto é, uma instância verdadeira da meta original é:

concatenar([a,b],[],[a,b])

Se for rejeitada esta solução, o "backtracking" gerará outras soluções:

X = [a] Y = [b]
X = [] Y = [a,b]

nesta ordem, recasando, com a segunda cláusula para 'concatenar', metas já resolvidas uma vez usando a primeira cláusula.

2.19.3. Características notáveis de programas lógicos

A simplicidade da sintaxe e da semântica de programas lógicos esconde várias características notáveis não encontradas em linguagens de programação convencionais. Segundo o trabalho de de Pereira e Warren, as características especialmente relevantes à escrita de gramática são:

(1) Casamento de padrões (unificação) substitui o uso convencional de funções de seletor e construtor para operações sobre dados estruturados.

(2) Os argumentos de um procedimento podem servir, não apenas para o mesmo receber um ou mais valores como entrada, mas também para retornar um ou mais valores como saída.

(3) Os argumentos de entrada e saída de um procedimento não têm que ser distingüidos antes, mas podem variar de uma chamada para outra.

(4) Os procedimentos podem gerar (via "backtracking", no caso do Prolog) um conjunto de resultados alternativos. Tais procedimentos são chamados "não-determinísticos". O "backtracking" é equivalente a uma forma de iteração de alto nível.

(5) Os procedimentos podem retornar resultados "incompletos", isto é, o termo ou termos retornados como resultado de um procedimentos podem conter variáveis, que são preenchidas apenas mais tarde, por chamadas de outros procedimentos. O efeito é similar ao uso de atribuição numa linguagem convencional para preencher campos de uma estrutura de dados. Note, entretanto, que podem existir muitas ocorrências de uma variável não instanciada e que todas elas são preenchidas simultaneamente (num único passo) quando a variável é finalmente instanciada. Note também que quando duas variáveis são unificadas, elas tornam-se uma única. O efeito é como um ponteiro invisível, ou referência, ligando uma variável à outra. Refere-se a este fenômeno como "variável lógica".

(6) O "programa" e os "dados" são idênticos na forma. Um procedimento consistindo somente de fatos, está mais perto de uma matriz, ou tabela de dados, em uma linguagem convencional.

Capítulo 3

O Sistema Proposto Baseado em Lógica e Conexionismo

3.1. Introdução

As várias abordagens existentes para o Processamento de Linguagem Natural (PLN) se traduzem em aplicação de técnicas de Inteligência Artificial, ou baseadas em lógica ou em sistemas conexionistas. Este trabalho faz um "mix" destas duas abordagens, acrescentando a extensão temporal da análise da sentença. O resultado é um sistema que faz a análise sintática, baseada na lógica de predicados do Prolog, as análises semântica e recorrente, baseadas numa abordagem de redes neurais, de frases da língua portuguesa. O sistema proposto possui léxico e regras de sintaxe limitados. Mas, para este universo, os resultados obtidos foram bastante satisfatórios.

3.2. O analisador sintático

O analisador sintático usa uma abordagem baseada em gramática de cláusulas definidas e é implementado em Prolog.

3.2.1. Usando Prolog

Com várias linguagens lógicas disponíveis, por que foi escolhido Prolog para construir a base de conhecimento, a estrutura sintático-semântica e a estrutura contextual? As razões são que Prolog é usado em sistemas de quinta geração e as linguagens são adaptáveis às necessidades do processamento de linguagem natural. No processamento de linguagem natural, três técnicas de programação são importantes (Geetha e Subramanian, 1990): dependência de contexto, construção de estrutura e a inclusão de condições extras para serem checadas. Pode-se facilmente incorporar essas características nos fatos e regras do Prolog.

O Prolog permite dependência de contexto em uma gramática: para saber que formas de uma frase são consideradas aceitáveis pode depender do contexto no qual a frase ocorre sintatica e semanticamente. Adicionando mais informação como argumentos nas definições de regras gramaticais, pode-se testar a informação contextual. Pode-se armazenar características como número, tempo verbal e gênero no léxico e usá-los para checar compatibilidade. Como as definições gramaticais são analisadas recursivamente, pode-se passar informação interpretada em qualquer ponto anterior ("backward") ou posterior ("forward") através da cadeia de interpretação.

O Prolog permite que se construa estruturas de árvores que possam fornecer uma representação da sintaxe e da semântica da cadeia de entrada. Pode-se realizar casamento de características usando o processo de unificação do Prolog.

Como Prolog é uma linguagem baseada em lógica, seus problemas são mais modulares e são baseados no formalismo simples e uniforme das cláusulas de Horn.

Prolog é especialmente adequada ao processamento de linguagem natural de várias formas. Pode-se escrever gramáticas de linguagem natural quase diretamente como programas Prolog, permitindo analisar e sintetizar sentenças em linguagem natural, executando diretamente os programas Prolog. Pode-se construir a estrutura sintática durante a análise, graças às propriedades de construção de estruturas do Prolog. Pode-se facilmente produzir a representação semântica de um texto em linguagem natural, usando a estrutura de "frame" baseada em lógica, por causa da conexão estreita existente entre Prolog e a lógica. Isto é também verdade quando constrói-se estruturas pragmáticas parciais (onde as sentenças são colocadas num contexto de conhecimento de fundo sobre palavras na sentença). Finalmente, o mecanismo de inferência do Prolog ajuda a representar inferências para respostas de questões.

3.2.2. Representando conhecimento em Prolog

Pode-se expressar conhecimento em Prolog, através de fatos e regras. As unidades básicas para a construção de fatos e regras são os **predicados**, isto é, expressões que dizem coisas simples sobre os indivíduos no universo. Por exemplo, a informação "Sonia ama Roberto" é representada por

ama (sonia,roberto).

Classes de indivíduos, que são introduzidas através das palavras "todos", "qualquer um", "alguém", etc., deve fazer uso de variáveis. Por exemplo, "todos são amados por sua mãe" poderia ser representado pela expressão da lógica

ama (mãe(x),x).

Os predicados são representados por um nome ("ama"), seguido de uma lista de argumentos. Cada argumento pode ser: (1) o nome de um indivíduo, também chamado de constante ("sonia", "roberto"); (2) uma variável ("x"); ou (3) uma expressão funcional ("mãe(x)"). As expressões funcionais podem ser imaginadas como nomes para esses indivíduos que estão na relação funcional correspondente com seus argumentos. Por exemplo, *mãe(x)* é o nome para um indivíduo que acontecer de ser a *mãe* de *x*. Pode-se representar um fato simplesmente escrevendo um predicado seguido de um ponto final:

ama (sonia,roberto).

O Prolog considerará isto como uma asserção que o predicado tem no universo que se está tentando descrever.

As regras, por outro lado, têm a forma geral

$$P_1 \text{ se } (P_2 \text{ e } P_3 \text{ e } \ldots \text{ e } P_n)$$

onde P_i significa os predicados e os parênteses geralmente são omitidos. Por exemplo, a regra geral que "Roberto ama todos que o amam" pode ser

ama(roberto,y) se ama(y,roberto).

Prolog terá uma regra estabelecendo que se todas as condições P_2, P_3,... P_n forem satisfeitas, então a conclusão P_1 será satisfeita. Se a regra contém variáveis, são aplicados todos os valores possíveis. Portanto, num universo onde os indivíduos são apenas Roberto e Sonia, a escrita da regra

ama(roberto,y) se ama(y,roberto)

levaria a escrever as duas regras

ama(roberto,roberto) se ama(roberto,roberto).
ama(roberto,sonia) se ama(sonia,roberto).

Estas regras são chamadas **instâncias** da regra mais geral. (Note que a primeira das duas regras, se aplicada tentando mostrar que Roberto ama ele mesmo, poderá levar a um "loop" infinito. A programação criativa tem mais efeito no Prolog do que em qualquer outra linguagem.)

Os indivíduos que constituem o universo são aqueles que são referidos nos fatos e regras, ou diretamente através de seus nomes de identificação (por exemplo, *roberto*), ou indiretamente através de expressões funcionais (por exemplo, *mãe(x)* pode denotar *mãe(roberto)*, *mãe(sonia)* ou *mãe(mãe(roberto))*, etc., num universo no qual as únicas constantes são *roberto* e *sonia*. Os fatos que contém variáveis devem aceitar todas as suas possíveis instâncias.

Dado o conhecimento sobre um certo domínio ou universo, tem-se usualmente meios alternativos de representá-lo através de fatos e regras. Por exemplo, a informação "todos são amados por sua mãe" poderia também ser representado

ama(x,y) se mãe(x,y)

(isto é, se *x* é a *mãe* de *y*, então *x ama y*). Aqui está-se usando um predicado binário ao invés de uma expressão funcional para representar o relacionamento "mãe" entre dois indivíduos. Qualquer

que seja a representação escolhida, deve-se manter-se consistente em todas as partes da descrição inteira do conhecimento. Cada indivíduo representado deve ter um único nome.

Pode-se representar mais informação com predicados do que com funções. Por exemplo, se se tiver "tio(x,y)" para significar que *x* é o *tio* de *y*, pode-se facilmente descrever um universo onde uma pessoa tem dois tios:

tio(tom,maria).
tio(josé,maria).

Se, entretanto, o único meio de referir-se ao tio de alguém for uma expressão funcional da forma tio(x), então pode-se apenas nomear um tio para cada indivíduo, pois tio(maria) denota um indivíduo único no universo.

3.2.3. Como escrever gramáticas na lógica

Vai-se descrever aqui o formalismo das gramáticas de cláusulas definidas.

3.2.3.1. Exprimindo gramáticas livres de contexto em cláusulas definidas

Para descrever como as gramáticas são expressas em lógica, começa-se considerando as gramáticas livres de contexto (GLCs). Para estas, usa-se a seguinte notação. Cada regra tem a forma:

nt → corpo.

onde *nt* é um símbolo não-terminal e *corpo* é uma seqüência de um ou mais itens separados por vírgulas. Cada item ou é um símbolo não-terminal ou uma seqüência de símbolos terminais. O significado da regra é que o corpo é uma forma possível para uma frase do tipo *nt*. Um símbolo não-terminal é escrito como um átomo do Prolog, enquanto que uma seqüência de terminais é escrito como uma lista do Prolog, onde um terminal pode ser qualquer termo do Prolog. A cadeia nula é escrita como a lista vazia '[]'. Como na sintaxe das cláusulas, esta notação básica é estendida

para permitir que alternativas apareçam no corpo. Seqüências alternativas de símbolos são separadas por ponto-e-vírgula, com parênteses onde necessário.

Vai-se mostrar agora uma GLC simples para ilustrar a notação (Pereira e Warren, 1980). A gramática cobre sentenças tais como "João ama Maria" e "Todo homem que vive ama uma mulher":

```
sentença --> sintagma_nominal, sintagma_verbal.
sintagma_nominal --> determinante, subst, cl_relativa.
sintagma_nominal --> nome.
sintagma_verbal --> verbo_trans, sintagma_nominal.
sintagma_verbal --> verbo_intrans.
cl_relativa --> [que], sintagma_verbal.
cl_relativa --> [].
determinante --> [todo].
determinante --> [uma].
subst --> [homem].
subst --> [mulher].
nome --> [joão].
nome --> [maria].
verbo_trans --> [ama].
verbo_intrans --> [vive].
```

Para conseguir a tradução, associa-se com cada não-terminal um predicado de dois argumentos (tendo o mesmo nome). Os argumentos do predicado representam os pontos de início e fim na cadeia de um sintagma para aquele não-terminal. As primeiras sete regras no exemplo ficam assim:

```
sentença(S0,S) :- sintagma_nominal(S0,S1), sintagma_verbal(S1,S2).
sintagma_nominal(S0,S) :- determinante(S0,S1), subst(S1,S2), cl_relativa(S2,S).
sintagma_nominal(S0,S) :- nome(S0,S).
sintagma_verbal(S0,S) :- verbo_trans(S0,S1), sintagma_nominal(S1,S).
sintagma_verbal(S0,S) :- verbo_intrans(S0,S).
cl_relativa(S0,S) :- conecta(S0,que,S1), sintagma_verbal(S1,S).
cl_relativa(S,S).
```

Pode-se ler a primeira cláusula como "uma sentença vai de S0 a S se existir um sintagma nominal de S0 a S1 e um sintagma verbal de S1 a S"; pode-se ler a última cláusula como "uma cláusula relativa que vai de S a S", ou seja, "uma cláusula relativa pode estar vazia".

Para representar os símbolos terminais em regras, usa-se um predicado de três argumentos, 'conecta', onde 'conecta(S1,T,S2)' significa "o símbolo terminal T está entre os pontos S1 e S2 na cadeia". Então as regras restantes são traduzidas em:

```
determinante(S0,S) :- conecta(S0,todo,S).
determinante(S0,S) :- conecta(S0,uma,S).
subst(S0,S) :- conecta(S0,homem,S).
subst(S0,S) :- conecta(S0,mulher,S).
nome(S0,S) :- conecta(S0,joão,S).
nome(S0,S) :- conecta(S0,maria,S).
verbo_trans(S0,S) :- conecta(S0,ama,S).
verbo_intrans(S0,S) :- conecta(S0,vive,S).
```

A primeira cláusula, por exemplo, lê-se "existe um determinante de S0 a S se a palavra *todo* estiver entre S0 e S".

Agora, para representar uma sentença particular para ser reconhecida, ou seja,

Todo homem que vive ama Maria.
1 2 3 4 5 6 7

rotula-se a sentença com inteiros, como mostrado, e a sentença é traduzida no seguinte conjunto de fatos:

```
conecta(1,todo,2).
conecta(2,homem,3).
conecta(3,que,4).
conecta(4,vive,5).
conecta(5,ama,6).
conecta(6,maria,7).
```

Então para determinar se aquela sentença é gramatical, tenta-se provar a meta:

```
?- sentença(1,7).
```

O procedimento de prova usado determina a estratégia de análise.

Note que a representação de uma gramática livre de contexto por cláusulas é **independente de dados**, no sentido que a representação real da cadeia a ser analisada não é "conhecida" pelas cláusulas - apenas o predicado "conecta" e a meta a ser provada leva isto em consideração. Se se rotula um ponto numa cadeia, não por um inteiro, mas por uma lista de símbolos ocorrendo depois daquele ponto na cadeia, não é mais necessário prover uma cláusula 'conecta' separada para cada símbolo na cadeia. Ao invés disto, pode-se definir o predicado 'conecta' numa cláusula simples mais geral:

```
conecta([W | S],W,S).
```

que pode ser lido como "A posição da cadeia rotulada pela lista com cabeça W e cauda S é conectada pelo símbolo W à posição da cadeia rotulada de S". A meta de provar a sentença original gramatical é agora expressa como:

```
?- sentença([todo,homem,que,vive,ama,maria],[]).
```

Dependendo do procedimento de prova, uma representação ou a outra pode ser preferida, por razões de eficiência. No caso do Prolog, as provas com as duas representações serão essencialmente a mesma, com rótulos de inteiros substituídos por segmentos finais da cadeia de entrada.

Note que nos casos onde a segunda representação é usada, é possível executar, ou "preprocessar", todas as chamadas ao 'conecta' em "tempo de compilação", por isto dispensando qualquer ajuda para referir ao predicado em "tempo de execução". Por exemplo, preprocessando desta forma a cláusula:

```
cl_relativa(S0,S) :- conecta(S0,que,S1), sintagma_verbal(S1,S).
```

ter-se-ia:

```
cl_relativa([que | S1],S) :- sintagma_verbal(S1,S).
```

Note que as regras gramaticais são diretamente identificadas com cláusulas definidas desta forma preprocessada e portanto não existe menção do predicado 'conecta'.

A seguir, discute-se como identificar as regras livres de contexto com cláusulas definidas de uma certa forma. Uma gramática livre de contexto é então identificada com um conjunto de tais cláusulas.

3.2.3.2. Gramáticas de cláusulas definidas

Sabe-se que as gramáticas de cláusulas definidas (GCDs) são uma extensão da gramática livre de contexto (GLC). Portanto, antes de discutir as GCDs, alguma coisa da GLC deve ser dita. Numa GLC, cada elemento da linguagem, do maior ao menor, deve ser definido separadamente.

As definições são feitas por "terminais" e "não-terminais". Um não-terminal é definido em termos de outros elementos da linguagem. Um terminal pode ser definido apenas em termos de si mesmo. Por exemplo, uma sentença pode ser formada de um sintagma nominal e de um sintagma verbal. O sintagma nominal é um não-terminal porque é definida em termos de um determinante seguido de um substantivo. O determinante é definido pelos terminais *o/a*, *um/uma* e *uns/umas*, que não são definidos em termos de nenhum outro.

Uma gramática é livre de contexto se cada elemento é independente dos outros - isto é, nenhuma parte da gramática influencia a forma como uma outra parte é analisada. Entretanto, as partes de algumas sentenças da linguagem podem influenciar-se de formas sutis. Por exemplo, o tempo verbal, pessoa e em muitos casos, o gênero das muitas partes de uma sentença da linguagem natural devem concordar para que a sentença seja gramaticalmente correta.

A maior desvantagem das GCDs é a excessiva quantidade de "backtracking" que pode ser gerado. Tomando cuidado com a definição da gramática, entretanto, pode-se assegurar que o Prolog analise a linguagem de forma mais eficiente. Deve-se usar adequadamente o *cut* (!), para reduzir ou até mesmo evitar o "backtracking" desnecessário.

Vai-se agora generalizar as gramáticas livres de contexto, duma forma que se manterá a correspondência com cláusulas definidas, para obter o formalismo das gramáticas de cláusulas definidas.

- Notação

A notação para as GCDs estende a notação para as GLCs na seguinte forma:

(1) É permitido que não-terminais sejam termos compostos em adição aos átomos simples permitidos no caso livre de contexto, por exemplo:

sn(X,S) sentença(S)

(2) No lado direito de uma regra, em adição aos não-terminais e listas de terminais, pode haver seqüências de chamadas de procedimentos, escritas dentro de chaves '{' e '}'. Elas são usadas para exprimir condições extras que devem ser satisfeitas para a regra ser válida, por exemplo,

subst(N) --> [W],{formaraiz(W,N),é_subst(N)}.

que pode ser lido como "a frase identificada como o substantivo N pode consistir da palavra simples W, onde N é a forma raiz de W e N é um substantivo".

Não-terminais, terminais e chamadas de procedimentos do lado direito de uma regra serão referenciados coletivamente como metas.

- O significado da notação GCD como cláusulas definidas

Os símbolos terminais são traduzidos exatamente como antes; um não-terminal de aridade N traduz em um predicado de N + 2 lugares (tendo o mesmo nome), cujos primeiros N argumentos são aqueles explícitos no não-terminal e cujos dois últimos são iguais aos da tradução de um não-terminal livre de contexto; as chamadas de procedimento do lado direito de uma regra são simplesmente traduzidas como elas mesmas. Por exemplo, a regra:

subst(N) --> [W], {formaraiz(W,N),é_subst(N)}

representa a cláusula:

```
subst(N,S0,S) :- conecta(S0,W,S), formaraiz(W,N), é_subst(N).
```

- O uso de gramáticas de cláusulas definidas

Vai-se discutir agora como o formalismo GCD provê três importantes mecanismos na análise de linguagem (Pereira e Warren, 1990): a construção de estruturas (tais como as árvores de análise), a imposição de condições extras nos constituintes de uma frase e um tratamento geral de dependência de contexto.

- Construindo estruturas

Os argumentos extras de não-terminais provêem os meios de construir estruturas nas regras gramaticais. Como os não-terminais são "expandidos", casando com as regras gramaticais, as estruturas são progressivamente construídas no curso do processo de unificação.

Vai-se apresentar aqui um exemplo simples. A gramática livre de contexto é modificada para produzir explicitamente para cada frase uma interpretação que é simplesmente sua árvore de análise. Vai-se introduzir uma forma mais compacta e mais eficiente de representar o **dicionário**, isto é, as regras definindo os não-terminais que correspondem às classes de palavras (ou partes do discurso). Em geral, ao invés de ter-se uma regra da forma:

```
categoria(argumentos) --> [palavra].
```

para cada *palavra* na classe *categoria*, escreve-se uma regra geral:

```
categoria(argumentos) --> [W], {cat(W,argumentos)}.
```

e define-se um "procedimento de dicionário" *cat* consistindo de cláusulas da forma:

```
cat(palavra,argumentos).
```

para cada *palavra* em *categoria*.

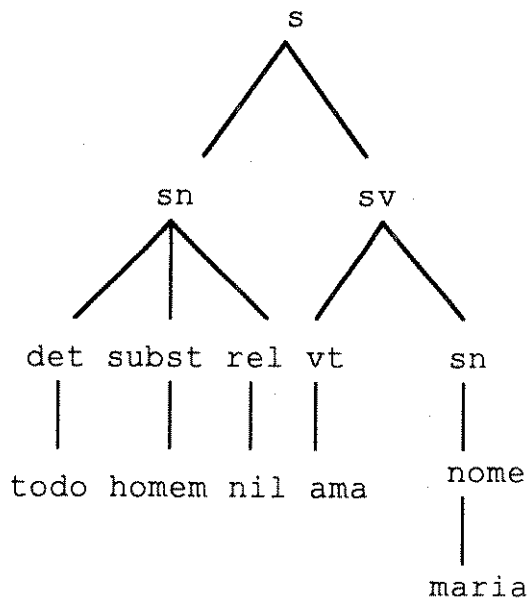
As regras para o exemplo modificado são:

```
sentença(s(SN,SV)) --> sintagma_nominal(SN), sintagma_verbal(SV).
sintagma_nominal(sn(Det,Subst,Rel))--> determinante(Det), subst(Subst), cl_relativa(Rel).
sintagma_nominal(sn(Nome)) --> nome(Nome).
sintagma_verbal(sv(VT,SN)) --> verbo_trans(VT), sintagma_nominal(SN).
sintagma_verbal(sv(VI)) --> verbo_intrans(VI).
cl_relativa(rel(que(SV)) --> [que], sintagma_verbal(SV).
cl_relativa(rel(nil)) --> [].
determinante(det(W)) --> [W], {é_determinante(W)}.
subst(subst(W)) --> [W], {é_subst(W)}.
nome(nome(W)) --> [W], {é_nome(W)}.
verbo_trans(vt(W)) --> [W], {é_trans(W)}.
verbo_intrans(vi(W)) --> [W], {é_intrans(W)}.
```

Lê-se um não-terminal aumentado tal como 'sintagma-nominal(SN)' como "um sintagma nominal com interpretação SN". Então a primeira regra é lida como "Uma sentença com interpretação s(SN,SV) pode consistir de um sintagma nominal com interpretação SN seguida por um sintagma verbal com interpretação SV". Exemplos de cláusulas a partir do dicionário associado são:

```
é_determinante(todo).
é_subst(homem).
é_nome(maria).
é_trans(ama).
é_intrans(vive).
```

A análise da sentença "Todo homem ama Maria" com estas regras produz a seguinte árvore de análise:



isto é, segue da semântica declarativa de cláusulas definidas que:

sentença(*theta*,[todo,homem,ama,maria],[])

é um termo verdadeiro, onde *theta* é o termo representado graficamente acima.

- Condições extras

O uso de chamadas explícitas de procedimentos no corpo da regra para restringir os constituintes aceitos, é ilustrado pela seguinte regra:

data(D,M) --> mês(M), [D], {inteiro(D), 0 < D, D < 32}.

Pode-se ler esta regra como "uma frase representando a data, dia D do mês M, pode ser escrita como uma frase representando o mês M seguido por um símbolo D, onde D é um inteiro maior que 0 e menor que 32".

- Dependência de contexto

Os argumentos de não-terminais numa GCD podem ser usados não apenas para construir estruturas mas também para carregar e testar informação contextual. Por exemplo, pode-se

modificar o exemplo anterior para trabalhar com o "número" (singular ou plural) necessário entre determinantes, substantivos e verbos. A gramática modificada aceitará sentenças como "Todo homem ama alguma garota" e "Todos os homens gostam de garotas", mas rejeitará uma sentença não-gramatical como "Todos os homens que vive amam uma mulher". Para trabalhar com número, certos não-terminais terão um argumento extra que pode ter os valores 'singular' ou 'plural'; os predicados do dicionário terão o argumento "número" e também um argumento para retornar a raiz de uma palavra. As regras modificadas são:

```
sentença(s(SN,SV)) --> sintagma_nominal(N,SN), sintagma_verbal(N,SV).
sintagma_nominal(N,sn(Det,Subst,Rel)) --> determinante(N,Det),
                                          subst(N,Subst), cl_relativa(N,Rel).
sintagma_nominal(singular,sn(Nome)) --> nome(Nome).
sintagma_verbal(N,sv(VT,SN)) --> verbo_trans(N,VT), sintagma_nominal(N1,SN).
sintagma_verbal(N,sv(VI)) --> verbo_intrans(N,VI).
cl_relativa(N,rel(que(SV)) --> [que], sintagma_verbal(N,SV).
cl_relativa(N,rel(nil)) --> [].
determinante(N,det(W)) --> [W], {é_determinante(W,N)}.
determinante(plural,det(nil)) --> [].
subst(N,n(Raiz)) --> [W], {é_subst(W,N,Raiz)}.
nome(nome(W)) --> [W], {é_nome(W)}.
verbo_trans(N,vt(Raiz)) --> [W], {é_trans(W,N,Raiz)}.
verbo_intrans(N,vi(Raiz)) --> [W], {é_intrans(W,N,Raiz)}.
```

Exemplos de cláusulas a partir do dicionário associado são:

```
é_determinante(todo,singular).
é_determinante(todos,plural).
é_subst(homem,singular,homem).
é_subst(homens,plural,homem).
é_nome(maria).
é_trans(ama,singular,ama).
é_trans(amam,plural,ama).
é_intrans(vive,singular,vive).
é_intrans(vivem,plural,vive).
```

- Como as GCDs são executadas pelo Prolog

Foram discutidas as GCDs a partir de um ponto de vista declarativo. Para entender a GCD isto é perfeitamente adequado - já que a GCD não é mais que um conjunto de cláusulas definidas, seu significado é independente de qualquer mecanismo de execução. Entretanto, como já foi notado, cada procedimento de prova para cláusulas definidas corresponde a uma estratégia de análise diferente para as GCDs. Discute-se agora o que isto significa no caso de um procedimento de prova do Prolog.

A partir da semântica procedimental do Prolog, segue que, para analisar uma sentença, as regras gramaticais são usadas "top-down", uma de cada vez, e as metas numa regra são executadas da esquerda para a direita (isto é, a sentença é analisada da esquerda para a direita). Se houverem regras alternativas em qualquer ponto, o "backtracking" eventualmente retornará a elas. É dever do elaborador da gramática formular a gramática de tal forma que o mesmo trabalho não seja repetido desnecessariamente em diferentes alternativas de "backtracking". Na prática isto não é tão difícil para linguagens que são lidas da esquerda para a direita, como o Português. Todo o trabalho de análise é feito pelo mesmo mecanismo uniforme (o procedimento de prova do Prolog) e o "backtracking" é realizado com muita eficiência.

Para mostrar como o Prolog executa uma GCD, e em particular o "backtracking" e o casamento de padrões, vai-se agora descrever os passos principais na análise da sentença (Pereira e Warren, 1980):

Aquele homem que assovia pinta paredes.
1 2 3 4 5 6 7

de acordo com a GCD na secção anterior, com um dicionário incluindo as seguintes cláusulas:

```
é_determinante(aquele,singular).
é_determinante(que,singular).
é_subst(homem,singular,homem).
é_subst(homens,plural,homem).
é_subst(pinta,singular,pinta).
é_subst(paredes,plural,parede).
é_trans(assovia,singular,assovia).
é_trans(pinta,singular,pinta).
```

é_intrans(assovia,singular,assovia).

Para maior legibilidade, vai-se escrever as metas do Prolog a partir de não-terminais ou terminais GCD na forma:

símbolo do ponto1 ao ponto2

onde símbolo é um não-terminal ou uma lista de terminais e ponto1, ponto2 são posições na cadeia de entrada, como rotulado acima.

A meta inicial é:

sentença(S) de 1 a 7

Isto casa com a regra simples para 'sentença', criando a instanciação:

S = s(SN,SV).

e as metas:

sintagma_nominal(N,SN) de 1 a P1,

sintagma_verbal(N,SV) de P1 a 7,

Prolog depois casa a primeira destas metas com a primeira regra para 'sintagma_nominal', produzindo a instanciação

SN = sn(Det,Subst,Rel).

e as metas:

determinante(N,Det) de 1 a P2,
subst(N,Subst) de P2 a P3,
cl_relativa(N,Rel) de P3 a P1,

Agora a primeira destas metas expande em duas submetas:

[W] de 1 a P2
é_determinante(W,N)

ambas as quais atingem o resultado desejado, já que a palavra na posição 1 da cadeia é "aquele" e porque o dicionário contém a cláusula:

é_determinante(aquele,singular).

A solução à meta 'determinante' é portanto:

determinante(singular,det(aquele)) de 1 a 2

Prolog agora continua com a meta para 'subst', que é correntemente instanciado a:

subst(singular,Subst) de 2 a P3

Isto é bem sucedido numa maneira similar à meta para 'determinante', com a solução:

subst(singular,n(homem)) de 2 a 3

Note que, se a palavra na posição 2 fosse "homens" ao invés de "homem", a meta não teria sido bem sucedida, já que nenhum casamento seria achado para a submeta intermediária:

é_subst(homens,singular,Raiz)

Prolog agora continua com a meta 'cl_relativa' casando com a primeira regra para 'cl_relativa', fornecendo as novas metas:

[que] de 3 a P4,
sintagma_verbal(singular,FV1) de P4 a P1,

A primeira destas metas tem sucesso com

P4 = 4

e a segunda casa com a primeira regra para 'sintagma_verbal', produzindo as submetas:

```
verbo_trans(singular,VT)          de 4 a P5,
sintagma_nominal(M,SN1)           de P5 a P1,
```

Ambas as submetas eventualmente têm sucesso, com soluções:

```
verbo_trans(singular,vt(assovia))      de 4 a 5
sintagma_nominal(singular,sn(det(nil),
                 subst(pinta),rel(nil))  de 5 a 6
```

onde a segunda solução é obtida via um casamento com a primeira regra para 'sintagma_nominal'.

Obteu-se agora uma solução para a meta 'sintagma_nominal' original, correspondendo a frase "aquele homem que assovia pinta". A próxima meta a ser executada, que vem a partir da ativação original da regra para 'sentença', é:

```
sintagma_verbal(singular,SV)      de 6 a 7.
```

(Lembre-se que a palavra na posição 6 é "paredes"). Casando esta meta com a primeira regra para 'sintagma_verbal' leva à meta:

```
verbo_trans(singular,VT1)         de 6 a P6,
```

que não pode ser bem sucedida pois "paredes" não é um 'verbo_trans'. Da mesma forma, a segunda regra para 'sintagma_verbal' falha, pois "paredes" não é um 'verbo_intrans'.

Neste ponto, Prolog faz o "backtracking" na mais recente meta para a qual ainda existe uma regra alternativa disponível, de nome:

```
sintagma_nominal(M,SN1)           de 5 a P1
```

Esta meta é agora casada com a segunda regra para 'sintagma_nominal', levando a meta:

```
nome(Nome)                        de 5 a P1
```

que falha porque a palavra na posição 5, "pinta", não é um 'nome'. Fazendo o "backtracking" novamente, a mais recente escolha é o uso da primeira regra para 'sintagma_verbal' para casar a meta:

sintagma_verbal(singular,SV1) de 4 a P1

Então, esta meta é agora casada com a segunda regra para 'sintagma_verbal', produzindo eventualmente a solução:

sintagma_verbal(singular,vi(assovia)) de 4 a 5

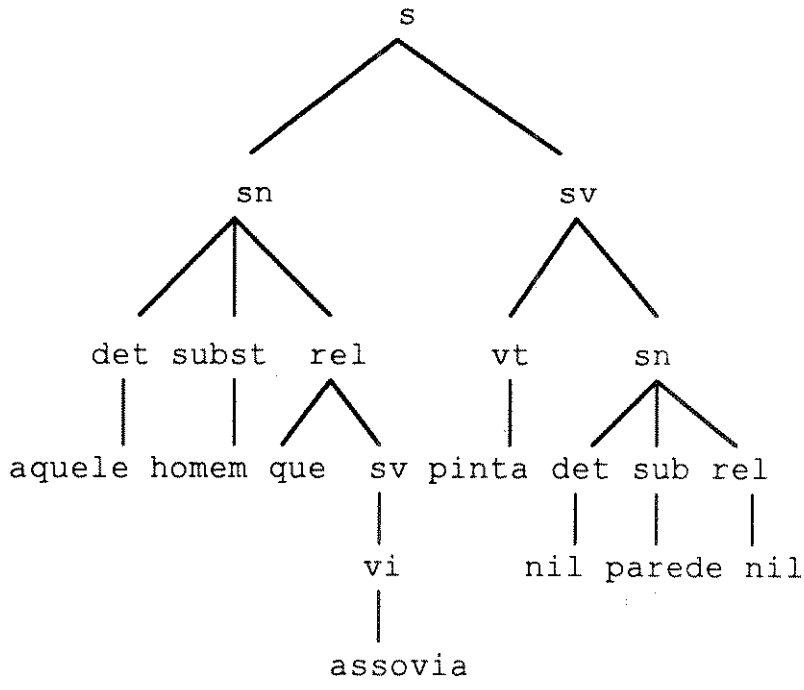
Achou-se agora uma segunda solução para a meta 'sintagma_nominal' original, correspondendo a frase "aquele homem que assovia". A única meta ainda pendente é a segunda meta da ativação original da regra para 'sentença'. Esta meta é correntemente instanciada a:

sintagma_verbal(singular,SV) de 5 a 7.

A execução da meta desta vez é bem sucedida, produzindo o resultado:

sintagma_verbal(singular,sv(vt(pinta),sn(det(nil),subst(parede),rel(nil))) de 5 a 7

completando então a análise. Colocando juntos as várias instanciações, obtém-se, como resultado S, a estrutura representada abaixo:



- O papel da variável lógica nas GCDs

A característica dos programas lógicos chamada de "variável lógica" torna as GCDs um formalismo muito poderoso para implementar práticos analisadores de linguagem. As estruturas podem ser construídas pedaço por pedaço, deixando partes não especificadas como variáveis. A estrutura pode ser passada e ser completada assim que a análise tem continuidade. Quando os fragmentos necessários estão disponíveis, os "buracos" na estrutura representados por variáveis são preenchidos pela unificação. Então é fácil construir termos com estruturas que não paralelizam a árvore de análise. Ilustra-se isto com um exemplo simples.

Neste exemplo, quer-se reconhecer sentenças comprendendo um verbo 'precede' ou 'segue' e nomes de meses, por exemplo, "Maio segue Abril". A interpretação dada a uma sentença deste tipo terá um termo da forma 'antes(M1,M2)'. Por exemplo, a interpretação de "Maio segue Abril" será 'antes(abril,maio)'. A gramática livre de contexto para este exemplo é dada pelas regras:

sentença --> mês,verbo,mês.
verbo --> [precede].
verbo --> [segue].
mês --> [janeiro].

```
mês --> [fevereiro].          etc.
```

Para construir a interpretação requerida, é dado os argumentos não-terminais extras com se segue:

```
sentença(S) --> mês(M1),verbo(M1,M2,S),mês(M2).
verbo(M1,M2,antes(M1,M2)) --> [precede].
verbo(M1,M2,antes(M2,M1)) --> [segue].
mês(janeiro) --> [janeiro].
mês(fevereiro) --> [fevereiro].
    etc.
```

Lê-se o não-terminal 'verbo(M1,M2,S)' como "um verbo cuja interpretação, no contexto de um sujeito M1 e objeto M2, é S".

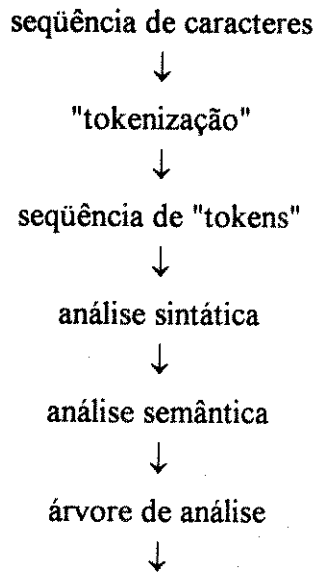
Note que, em geral, não é necessário que os primeiros dois argumentos de 'verbo' sejam conhecidos quando o procedimento de prova constrói o terceiro argumento durante a execução de uma regra para 'verbo'. Isto é, o relacionamento entre os argumentos de um não-terminal é definido independentemente de qualquer ordem particular de execução das regras.

Executando este exemplo com o Prolog, a análise dá-se da esquerda para a direita, de tal forma que na primeira regra, a estrutura S é construída antes de um de seus componentes, M2, ser conhecido. Este componente é preenchido mais tarde durante a análise do resto da sentença. Note que é oneroso e pouco natural deixar a construção da estrutura S para depois, até que M2 seja conhecido, isto é, até o fim da regra para 'sentença', já que a forma de S depende crucialmente da natureza do verbo.

3.2.4. Técnicas para Processamento de Linguagem

Não importa com que linguagem se trabalhe, os mesmos passos são aplicados para entender os comandos. Compiladores, interpretadores e processadores de comandos devem separar uma cadeia de letras em palavras, devem aplicar regras para reconhecer comandos válidos na linguagem e possuir métodos para descobrir o significado destes comandos. Estes passos são

chamados de "tokenização"[10], "análise sintática" e "análise semântica". Cada estágio tem entradas e resultados bem definidos, como mostra a figura abaixo (Marcus, 1986):



Um "token" é o menor objeto com significado na linguagem, tal como a palavra. Átomos, inteiros, operadores e variáveis são os "tokens" do Prolog. Com estes "tokens" , os termos maiores, como as estruturas e cláusulas, podem ser construídos.

Quando o usuário usa o teclado para entrar com sua sentença, o computador lê esta sentença como uma seqüência, ou lista, de códigos ASCII de caracteres. A "tokenização" é o processo de achar os "tokens" na lista de caracteres. O "tokenizador" converte a lista de caracteres em uma lista de "tokens", que é então passada para o analisador sintático e semântico.

Durante a análise sintática e semântica, o processador de linguagem produz uma representação intermediária do comando. Ele agrupa as partes do comando em uma estrutura de árvore, ou árvore de análise. Prolog provê uma notação conveniente para definir as "gramáticas de cláusulas definidas" ou GCDs.

[10] Vai-se usar aqui a palavra "token" e suas derivadas "tokenizar", "tokenização", etc., para simplificar o entendimento, muito embora estas palavras não existam na língua portuguesa.

3.2.4.1. "Tokenização"

A "tokenização" é o primeiro estágio no processo de entendimento de linguagem. O computador deve converter uma seqüência de caracteres em uma seqüência, ou lista, de "tokens". A cadeia de caracteres é uma lista entre aspas e é tratada como uma lista de códigos ASCII.

O fim de uma expressão é marcado pela lista vazia.

O "tokenizador" é um predicado recursivo que age sobre a cabeça de uma lista de caracteres e passa a cauda à próxima iteração (Marcus). Cada iteração olha a cabeça da lista para determinar se encontrou um caractere, um "token", um espaço ou o fim da lista. O primeiro argumento para o "tokenizador", portanto, é aquela parte da lista de entrada que ainda não foi "tokenizada".

Em seu segundo argumento, o "tokenizador" mantém uma lista de "tokens" que foram encontrados. É usado um predicado *append*, que deve ter um argumento de variável que não se torna instanciado até a última iteração do loop, quando um predicado recursivo retorna um valor. Da mesma forma, o terceiro argumento ao "tokenizador" permanece não instanciado até o último "token" ser adicionado à lista.

A primeira cláusula "tokenizar" tem sucesso quando a cabeça da lista contém uma letra. Neste caso, o "tokenizador" sabe que foi encontrada uma palavra. Ele passa então a lista de entrada e o primeiro caractere da palavra para *pega_resto_pal* para "tokenizar" o resto da palavra. Quando *pega_resto_pal* retorna, "tokenizar" coloca no final da lista a letra e "tokeniza" o resto da lista de entrada.

As letras maiúsculas são convertidas para minúsculas para que não haja confusão se o "token" é um átomo ou uma variável. O "tokenizador" pode checar as letras com as seguintes regras:

```
letra(C,D) :-
        C >= 'A, C =< 'Z,
        D is C + 32.
letra(C,C) :-
        C >= 'a, C =< 'z.
```

```
append([],L,L).
append([H|T],L,[H|T1]) :- append(T,L,T1).

tokenizar([H|T],Lista,L) :-
        letra(H),!,
        pega_resto_pal(T,[H],Pal,Res),
        append(Lista,[Pal],NovaLista),
        tokenizar(Res,NovaLista,L).
```

Uma palavra pode terminar com um espaço (código ASCII 32) ou com um caractere de pontuação como a vírgula, dois-pontos, ponto-e-vírgula, hífen ou ponto final. Eles são tratados como "tokens" de um único caractere. Estes "tokens" podem ser reconhecidos pelos seguintes fatos:

```
token(',,,).
token(';,;).
token(':,:).
token('-,-).
token('.,.).
```

Quando um operador é encontrado na cabeça da lista, a segunda cláusula "tokenizar" tem sucesso. Ele coloca o "token" no final da lista de "tokens" e chama "tokenizar" novamente com o restante da lista de entrada.

```
tokenizar([H|T],Lista,L) :-
        token(H,Token), !,
        append(Lista,[Token],NovaLista),
        tokenizar(T,NovaLista,L).
```

"Tokenizar" salta sobre os espaços (caractere ASCII 32). Note que "tokenizar" não detecta o primeiro espaço que separa os "tokens"; estes são encontrados por *pega_rest_pal*. "Tokenizar" localiza apenas os espaços extras.

```
tokenizar([32|T],Lista,L) :-
        !, tokenizar(T,Lista,L).
```

A lista vazia marca o fim da cadeia de caracteres. Quando a lista vazia é detectada, a "tokenização" é completada e o terceiro argumento torna-se instanciado.

```
tokenizar([],Lista,Lista).
```

Vai-se observar agora a forma como o "tokenizador" trabalha com palavras. Quando "tokenizar" detecta que encontrou uma letra, ele passa a primeira letra e o resto da lista para *pega_resto_pal*. *Pega_resto_pal* coleta as letras que fazem uma palavra e então converte aqueles caracteres para uma palavra. Isto é feito pelo cheque recursivo do caractere no início da lista. Se o caractere é uma letra, então a letra é colocada no final do segundo argumento. *Pega_rest_pal* então chama a si mesmo com o resto da lista de entrada e a coleta de letras continua.

```
pega_resto_pal([H|T],Lista,Pal,X) :-
        letra(H), !,
        append(Lista,[H],Plista),
        pega_resto_pal(T,Plista,Pal,X).
```

A palavra termina com um espaço ou com um "token" (caractere de pontuação). Quando ele encontra uma dessas condições, *pega_resto_pal* sabe que alcançou o fim da palavra. Então converte a lista de caracteres na palavra e retorna a lista de caracteres restante. A lista de caracteres ASCII é convertida a uma palavra por um predicado chamado *name*. Este predicado converte listas a átomos e átomos a listas. Note que o "token" não é removido da lista e então ele pode ser manipulado pelo predicado "tokenizar".

```
pega_resto_pal([32|T],Lista,Pal,T) :-
        name(Pal,Lista), !,
pega_resto_pal([H|T],Lista,Pal,[H|T]) :-
        token(H,_),
        name(Pal,Lista), !.
```

Se *pega_resto_pal* encontra a lista vazia, então ele sabe que alcançou tanto o fim da palavra como o fim da lista de entrada. Neste caso, ele converte a lista de letras na palavra e retorna a lista vazia ao "tokenizar".

```
pega_resto_pal([],Lista,Pal,[]) :-
        !, name(Pal,Lista).
```

Outra cláusula checa se o primeiro caractere da palavra é uma letra.

```
tokenizar([H|T],Lista,L) :-
        letra(H,Letra), !,
        pega_resto_pal(T,[Letra],Palavra,Res),
        append(Lista,[Palavra],NovaLista),
        tokenizar(Res,NovaLista,L).
```

O predicado *pega_resto_pal* coleta as letras da cadeia de entrada até achar um espaço ou ponto final. Então, *name* converte a lista em um átomo.

```
pega_resto_pal([H|T],Lista,Palavra,X) :-
        letra(H,Id), !,
        append(Lista[Id],Nlista),
        pega_resto_pal(T,Nlista,Palavra,X).
pega_resto_pal([32|T],Lista,Palavra,T) :-
        name(Palavra,Lista), !.
pega_resto_pal(['.|T],Lista,Palavra,T) :-
        name(Palavra,Lista), !.
pega_resto_pal([],Lista,Palavra,[]) :-
        !, name(Palavra,Lista).
```

3.3. Os analisadores semântico e recorrente

O analisador semântico foi implementado usando o conceito de microcaracterística semântica de Waltz e Pollack (1985) e McClelland e Kawamoto (1986). O analisador recorrente considerou como base o trabalho de Elman (1990). Ambos utilizam o algorítmo conexionista "backpropagation".

3.3.1. Contexto: Introdução

Existe problemas ao se usar nós "estabelecedores de contexto" como uma solução ao problema da interpretação da linguagem direcionada ao contexto. Uma determinada palavra estabelecedora de contexto, por exemplo, "caça", pode nunca ter sido explicitamente mencionada no texto ou discurso, mas pode ser facilmente inferida por um leitor ou ouvinte. Por exemplo, a sentença

Pedro passou o fim de semana na floresta.

poderia ser suficiente para induzir o contexto de "caça" ou a menção de palavras, ou frases, como "ao ar livre", "caminhada", "acampamento", etc., que são palavras estreitamente relacionadas com muitos outros conceitos além de "caça". Pode-se concluir (a) a necessidade de inferir o conceito especial estabelecedor de contexto "caça", dada qualquer palavra ou frase acima; ou (b) a necessidade de providenciar as conexões entre cada uma das palavras ou frases e todos os sentidos possíveis destas palavras ou frases.

Propõe-se que cada conceito deva ser representado não meramente como um nó unitário, mas também como a associação desse nó com um conjunto de "microcaracterísticas" que servem para (a) definir os conceitos, ao menos parcialmente, e (b) associar o conceito com outros que compartilham suas microcaracterísticas. Propõe-se um conjunto grande de microcaracterísticas, cada uma potencialmente conectada com todo nó de conceito no sistema proposto. Cada conceito é de fato conectado a apenas algum subconjunto do conjunto total, via ligações de, ou ativação ou inibição, bidirecional. Conceitos estreitamente relacionados têm muitas microcaracterísticas em comum. As microcaracterísticas são parte de um módulo que pode ser dirigido pela percepção, entrada de linguagem e memória.

Sugere-se que as microcaracterísticas devam ser escolhidas tomando por base os primeiros princípios que correspondem às maiores distinções que as pessoas fazem a respeito das situações no universo. Por exemplo (Waltz e Pollack, 1985), algumas microcaracterísticas importantes correspondem a distinções tais como *perigoso/seguro*, *animado/inanimado*, *comestível/não-comestível*, *em movimento/parado*, *intencional/sem intenção*, comprimentos característicos de eventos (por exemplo, eventos de tempo requerem milisegundos, horas ou anos), locações (determinadas cidades, países, continentes, etc.) e tempos históricos (datas ou periodos). Como no

modelo de Hinton (1981), as hierarquias surgem naturalmente, baseadas nos subconjuntos de microcaracterísticas compartilhadas, mas não são a base fundamental para a organização de conceitos numa rede semântica, como em muitos modelos de Inteligência Artificial.

3.3.1.1. Microcaracterísticas e Contexto

Idealmente, o conjunto particular de microcaracterísticas associadas com um conceito deveria servir a dois propósitos: (1) deveria ser suficiente para distingüir o conceito de todos os outros e (2) deveria ter compartilhado microcaracterísticas com todos os conceitos que estão associados com o conceito dado, mas que não estejam relacionados a eles nas formas que geralmente classifica-se como "associações livres" comuns ou relações n-árias. O conjunto de microcaracterísticas é então parcialmente definicional, mas estritamente falando, não existe uma definição completa para um conceito neste modelo. Ou seja, os conceito são definidos por suas posições em uma rede, isto é, as coisas às quais eles estão conectados.

Representou-se todas as microcaracterísticas possíveis como um vetor, onde cada posição do vetor corresponde a uma microcaracterística independente e o valor numérico dessa posição corresponde ao nível de ativação dessa característica. Pode-se imaginar que algumas microcaracterísticas devam ser conectadas diretamente às outras por ligações mutuamente inibitórias ou ligações mutuamente excitatórias.

Apenas um subconjunto de combinações possíveis de valores de microcaracterísticas pode ocorrer como contextos; ainda que um sistema perceptual poderia, em princípio, induzir quaisquer valores para as microcaracterísticas num sistema completo, o mundo real de fato comporta-se de uma maneira ordenada, tal que apenas certas combinações de valores poderiam realmente ser observadas.

Por causa das restrições de memória e perceptual, não seria possível ter mais que um pequeno número de estabelecedores de contexto representados duma vez no vetor. Dois ou mais contextos seriam necessários para entender a interação de uma pessoa com uma outra, um para cada visão de mundo que inclui a outra pessoa. Se a visão que a outra pessoa tem da primeira é bem conhecida e suficientemente diferente da visão que a primeira tem dela própria, seriam

necessários, no mínimo, três contextos: (1) o da primeira pessoa, (2) o da outra pessoa e (3) o modelo que a outra pessoa faz da primeira[11].

3.3.2. Mecanismos do Processamento da Sentença

Como muitos processos cognitivos naturais, o processo da compreensão de sentença envolve a consideração simultânea de muitas fontes diferentes de informação. Vai-se considerar um aspecto da compreensão da sentença, que é a atribuição dos constituintes de uma sentença aos casos temáticos corretos. A atribuição de casos reflete um aspecto importante do processo da compreensão, ou seja, a especificação de quem fez o quê para quem.

A atribuição de caso não é simples, como pode-se ver considerando algumas sentenças e os casos que se atribui aos seus constituintes. Considere, inicialmente, as seguintes sentenças usando o verbo *quebrar* (McClelland e Kawamoto, 1986):

(1) O garoto quebrou a vidraça.
(2) A pedra quebrou a vidraça.
(3) A vidraça quebrou.
(4) O garoto quebrou a vidraça com a pedra.
(5) O garoto quebrou a vidraça com a cortina.

Pode-se ver que a atribuição de casos aqui é bem complexa. O primeiro sintagma nominal (SN) da sentença pode ser o Agente (sentenças 1, 4 e 5), o Instrumento (sentença 2) ou o Paciente (sentença 3). O SN no sintagma preposicional (SP) poderia ser o Instrumento (sentença 4) ou poderia ser um Modificador do segundo SN, ou como é, no mínimo, uma leitura da sentença 5. Um outro exemplo traz a ambigüidade da atribuição de caso dos com-SNs:

(6) O garoto comeu o macarrão com molho.
(7) O garoto comeu o macarrão com o garfo.

---

[11] Quando crianças, pode-se suportar apenas um único contexto: o egocêntrico. O desenvolvimento da habilidade de simultaneamente suportar mais de um contexto vem mais tarde e pode ser o resultado de dividir o vetor de contexto em dois subconjuntos, cada um razoavelmente completo, mas que (1) pode ser separadamente ativado como um todo e (2) pode suportar diferentes padrões de ativação. Os mecanismos para que isto ocorra podem aumentar gradualmente, por processos tais que convertam grupos de microcaracterísticas em coisas concretas que freqüentemente ocorrem simultaneamente.

Em (6) o com-SN claramente não especifica um Instrumento, mas em (7) sim.

O significado das palavras nestas sentenças influencia a atribuição de argumentos a casos. Entretanto, a colocação dos SNs dentro das sentenças é também muito importante. Considere estas duas frases:

(8) O vaso quebrou a vidraça.
(9) A vidraça quebrou o vaso.

Aqui, deve-se depender das restrições da ordem de palavra. As restrições da ordem de palavra são muito fortes em português, mas é importante imaginar que tal dependência em tais restrições não é universal. A atribuição de caso é influenciada por pelo menos três diferentes tipos de fatores: ordem da palavra, restrições semânticas e (quando disponível) morfologia inflexional. Em adição a esses fatores, existe mais um que não pode ser ignorado, o contexto mais global no qual a sentença está inserida. Considere, por exemplo, a sentença 10:

(10) O garoto viu a menina com os binóculos.

Tem-se uma leitura se um contexto anterior informar que "um garoto estava olhando através da janela, tentando descobrir quanto ele poderia ver com vários instrumentos ópticos". Uma outra leitura seria possível se houvesse a informação anterior que "duas meninas estavam tentando identificar alguns pássaros quando um garoto chegou. Uma menina tinha um par de binóculos e a outra não".

Enquanto o fato de que a ordem da palavra e as restrições semânticas influenciam a atribuição de caso tem sido reconhecido, existem alguns poucos modelos que vão além e propõem um mecanismo para explicar a causa destes efeitos. Entretanto, existem alguns pesquisadores em processamento de linguagem que têm tentado encontrar formas de trazer as considerações semânticas para o processamento sintático de uma forma ou outra. Uma abordagem recente depende do léxico para influenciar o processamento sintático e a construção de representações funcionais básicas, que consideram casos como:

(11) A mulher queria o vestido do armário.
(12) A mulher colocou o vestido no armário.

A leitura preferida para a primeira destas sentenças tinha *do armário* como um modificador de *o vestido*, enquanto que a leitura preferida para a segunda tinha *no armário* como um argumento de local de *colocou*. Para explicar esta diferença na atribuição de caso, foi proposto dois princípios: (a) **preferência léxica** e (b) **argumentos finais**. Basicamente, a preferência léxica estabelece uma estrutura de argumento esperada (por ex. Sujeito-Verbo-Objeto no caso de *querer*; Sujeito-Verbo-Objeto-Objeto Preposicional no caso de *colocar*) consultando uma lista ordenada de possíveis estruturas de argumentos associadas com cada verbo. Se um constituinte que poderia preencher uma posição na estrutura de argumento esperada é encontrado, o mesmo é tratado como um argumento do verbo. Portanto, se um constituinte que aparece para satisfazer as condições do argumento final da estrutura de argumento esperada é encontrado, sua colocação na sentença é atrasada para permitir a incorporação nos constituintes dos constituintes subseqüentes. Portanto, com *querer*, o SN *o vestido* é um candidato para argumento final e não é colocado diretamente como um constituinte do Sintagma Verbal (SV); antes, uma estrutura SN superordenada contendo *o vestido do armário* é finalmente colocada na SV. Com *colocar*, entretanto, *o vestido* não poderia ser o argumento final, e portanto, ele é colocado diretamente ao SV. *No armário* está então disponível para a colocação como argumento final do SV.

De qualquer forma, está claro que um mecanismo é necessário no qual todos os constituintes de uma sentença possam trabalhar simultaneamente para influenciar a atribuição de casos aos constituintes. Este modelo toma como entrada uma análise parcial superficial e gera a partir disto uma representação de nível de caso.

3.3.2.1. Metas

A meta primária do modelo proposto é prover um mecanismo que possa começar a considerar conjuntamente o caso da ordem da palavra e restrições semânticas na atribuição do caso. Deseja-se que o modelo seja capaz de **aprender** a fazer isto baseado em experiência com sentenças e suas representações de caso. Deseja-se que este modelo seja capaz de **generalizar** o que aprendeu para novas sentenças formadas de combinações de palavras.

Em adição, tem-se outras metas para o modelo:

. Deseja-se que o modelo seja capaz de selecionar contextualmente leituras apropriadas de palavras ambíguas.

. Deseja-se que o modelo selecione o "frame" de verbo apropriado baseado nos padrões de argumentos e de suas características semânticas.

. Deseja-se que o modelo preencha os argumentos ausentes nas sentenças incompletas com valores "default" plausíveis.

. Deseja-se que o modelo seja capaz de generalizar seu conhecimento de atribuição de caso correto a sentenças que contêm uma palavra nunca vista antes, dada apenas uma especificação de algumas das propriedades semânticas da palavra.

O modelo está, óbviamente, muito longe de um modelo completo do processamento de sentença ou mesmo da atribuição de casos. O modelo não "resolve o problema da compreensão de sentença", mas sugere novas formas de pensar sobre muitos aspectos da linguagem e da representação da linguagem.

3.3.2.2. A arquitetura do modelo

O modelo consiste de dois conjuntos de unidades: um para representar a estrutura superficial da sentença e um para representar sua estrutura de caso. O modelo aprende através de apresentações de pares corretos de estrutura superficial e estrutura de caso; durante o teste, é apresentada a entrada em estrutura superficial e examina-se a saída que o modelo gera no nível de estrutura de caso.

- Sentenças

As sentenças processadas pelo modelo consistem de um verbo e de um a três SNs. Existe sempre um SN Sujeito e opcionalmente pode existir um SN Objeto. Se este estiver presente, pode também existir um com-SN; isto é, um SN em um sintagma preposicional de final de sentença começando com a palavra *com*.

- Formato de entrada das sentenças

O que o modelo vê como entrada não é a sentença propriamente dita, mas uma representação canônica da estrutura constituinte da sentença, na forma que poderia ser produzida por um simples analisador de superfície e um simples léxico.

3.3.2.3. Microcaracterísticas semânticas

Nos formatos de entrada canônicos, as palavras são representadas como listas de microcaracterísticas semânticas (Waltz e Pollack, 1985; McClelland e Kawamoto, 1986). Para substantivos e verbos, as características são agrupadas em muitas dimensões. Cada dimensão consiste de um conjunto de valores mutuamente exclusivos e, em geral, cada palavra é representada por um vetor no qual um, e apenas um, valor em cada dimensão está "ON" (ligado) para a palavra e todos os outros valores estão "OFF" (desligados). Valores que estão "ON" são representados nos vetores de características como "1"s. Valores que estão "OFF" são representados por pontos (".").

Segundo o trabalho de McClelland e Kawamoto, foram escolhidas as dimensões e os valores em cada dimensão para capturar o que se considerou dimensões importantes de variações semânticas nos significados de palavras que tinham implicações para a atribuição de casos das palavras.

O conjunto completo das dimensões usadas nos conjuntos de características é dado na tabela 3.1. As dimensões dos substantivos são auto-explicativas. Já em relação às dimensões dos verbos, há a necessidade de algum esclarecimento. A dimensão REALIZADOR indica se existe um Agente instigando o evento. A dimensão CAUSA especifica se o verbo é causal. Se não, é porque não existe causa especificada (como no caso de a *vidraça quebrou*) ou é porque não há troca (como no caso de o *garoto tocou a menina*). A dimensão TOQUE indica se o Agente, o Instrumento, ambos ou nenhum toca o Paciente. A dimensão NAT_TROCA especifica a natureza da troca que tem lugar no Paciente. A AGT_MVMT e a PT_MVMT especificam o movimento do Agente e do Paciente, respectivamente; e INTENSIDADE simplesmente indica a força da ação. Os rótulos dados às dimensões são, é claro, apenas para referência; eles são escolhidos de tal forma que cada dimensão de substantivo ou verbo tenha uma única primeira letra que possa ser usada para designar a dimensão.

Deve-se enfatizar que outras características podem ser incluídas para estender o modelo para conjuntos maiores de substantivos e verbos. As características incluídas foram cuidadosamente escolhidas porque elas parecem relevantes para determinar as atribuições de casos. Por exemplo, a dimensão REALIZADOR diretamente especifica se existe ou não um Agente. Portanto, as características do verbo, em particular, freqüentemente têm implicações caso-estruturais diretas.

As figuras 3.1 e 3.2 dão os vetores que se atribui a cada palavra usada no modelo. Algumas dessas decisões são arbitrárias e algumas são forçadas. As *bolas* são *redondas*, mas podem ser *moles* ou *duras*, etc. Ver-se-á que o modelo tende a corrigir esta deficiência.

Umas das metas para o modelo é mostrar como ele pode selecionar o significado apropriado no contexto para uma palavra ambígua. Para palavras ambíguas (*galinha*, viva ou cozida) o padrão de entrada é a média dos padrões de características de cada uma das duas leituras da palavra. Isto significa que nos casos onde as duas concordam com o valor de uma dimensão de entrada particular, esta dimensão tem o valor acordado na representação de entrada. Nos casos onde os dois discordam, a característica tem o valor de .5 (representado por "?") na representação de entrada. Um dos objetivos das simulações é ver se o modelo pode corretamente preencher estes valores não especificados, efetivamente recuperando os valores perdidos do contexto no processo da atribuição da palavra ao caso apropriado. A figura 3.1 indica as duas leituras de *galinha*, assim como as formas ambíguas usadas como entradas[12].

[12] Para o conceito *alimento* que é tido como o paciente implícito em sentenças como *o garoto comeu*, nenhuma forma particular parece apropriada. Portanto, a representação de saída pretendida é assumida não especificada (como indicada por "?") para todos os valores da dimensão "forma". Para todas as outras dimensões, *alimento* tem os valores típicos para alimentos.

TABELA 3.1

SUBSTANTIVOS - DIMENSÕES	VALORES DE CARACTERÍSTICAS
HUMANO	humano, não humano
SUAVIDADE	mole, duro
GÊNERO	masculino, feminino
VOLUME	pequeno, médio, grande
FORMA	1-D, 2-D, 3-D
PONTA	pontiagudo, redondo
DUREZA	frágil, inquebrável
TIPO-OBJ	alimento, brinquedo, ferramenta/utensílio, animado

VERBOS - DIMENSÕES	VALORES DE CARACTERÍSTICAS
REALIZADOR	sim, não
CAUSA	sim, não
TOQUE	agente, instrumento, ambos, nenhum
NAT_TROCA	peças, fragmentos, química, nenhum
AGT_MVMT	trans, part, nenhum
PT_MVMT	trans, part, nenhum
INTENSIDADE	baixo, alto

SUBSTANTIVOS	HU	SU	GE	VOL	FOR	PO	DU	TIP
alimento	.1	1.	1.	1..	???	.1	1.	1...
batata	.1	1.	.1	1..	1..	.1	1.	1...
bola	.1	1.	.1	1..	1..	.1	.1	.1..
boneca	.1	1.	.1	1..	..1	.1	1.	.1..
cachorro	.1	1.	1.	.1.	..1	.1	.1	...1
cenoura	.1	.1	.1	1..	1..	1.	1.	1...
colher	.1	.1	.1	1..	1..	.1	.1	..1.
cortina	.1	1.	.1	.1.	.1.	.1	1.	..1.
escrivaninha	.1	.1	.1	..1	..1	1.	.1	..1.
galinha	.1	1.	.1	1..	..?	.1	??	?..?
galinha cozida	.1	1.	.1	1..	1..	.1	1.	1...
galinha viva	.1	1.	.1	1..	..1	.1	1.	...1
garfo	.1	.1	1.	1..	1..	1.	.1	..1.
garoto	1.	1.	1.	.1.	..1	.1	.1	...1
homem	1.	1.	1.	..1	..1	.1	.1	...1
leão	.1	1.	1.	..1	..1	1.	.1	...1
lobo	.1	1.	1.	.1.	..1	1.	.1	...1
macaco	.1	??	1.	1..	..1	1.	.1	..??
macaco ferram.	.1	.1	1.	1..	..1	1.	.1	..1.
macaco animal	.1	1.	1.	1..	..1	1.	.1	...1
macarrão	.1	1.	1.	1..	1..	.1	1.	1...
martelo	.1	.1	1.	1..	1..	.1	.1	..1.
menina	1.	1.	.1	.1.	..1	.1	.1	...1
mulher	1.	1.	.1	..1	..1	.1	.1	...1
ovelha	.1	1.	.1	.1.	..1	.1	1.	...1
pedra	.1	.1	.1	1..	..1	1.	.1	..1.
prato	.1	.1	1.	1..	.1.	.1	1.	..1.
queijo	.1	1.	1.	1..	.1.	.1	1.	1...
vaso	.1	.1	1.	1..	1..	.1	1.	..1.
vidraça	.1	.1	.1	.1.	.1.	1.	1.	..1.

Figura 3.1. Os substantivos usados no modelo e suas microcaracterísticas semânticas (McClelland e Kawamoto, 1986)

VERBO	RE	CA	TOQU	N_TR	A_M	P_M	IN
bateu	1.	.1	.1..	...1	.1.	..1	.1
bateuAVPI	1.	.1	.1..	...1	.1.	..1	.1
bateuAVP	1.	.1	1...	...1	.1.	..1	.1
bateuIVP	.1	.1	.1..	...1	..1	..1	.1
comeu	1.	1.	..1.	..1.	.1.	1..	1.
comeuAVP	1.	1.	..1	..1.	.1.	1..	1.
comeuAVPI	1.	1.	..1	..1.	.1.	1..	1.
comeuAVF	1.	1.	..1.	..1.	.1.	1..	1.
moveu	1.	1.	1...	...1	1..	1..	1.
moveuAVP	1.	1.	1...	...1	1..	1..	1.
moveuAVS	1.	1.	...1	...1	1..	1..	1.
quebrou	1.	1.	.1..	1...	.1.	..1	.1
quebrouAVPI	1.	1.	.1..	1...	.1.	..1	.1
quebrouAVP	1.	1.	1...	1...	.1.	..1	.1
quebrouIVP	1.	.1	.1..	1...	..1	..1	.1
quebrouPV	1.	.1	...1	1...	.1.	..1	.1

Figura 3.2. Os verbos usados no modelo e suas representações de microcaracterísticas. As formas, seguidas por cadeias de letras maiúsculas representam os padrões de características alternativas entre os quais o modelo pode escolher, para especificar a leitura apropriada do verbo no contexto. Estes padrões de características alternativas correspondem às características semânticas do verbo apropriado para configurações particulares de casos, como indicado pelas letras: A = Agente, V = Verbo, P = Paciente, I = Instrumento, M = Modificador, S = Auto ("Self"), F = Alimento implícito. A posição da letra indica a posição do constituinte correspondente na sentença de entrada. Os padrões dados com o verbo genérico (sem as letras) são usados nas representações de entrada, do nível de sentença (McClelland e Kawamoto, 1986).

Uma outra meta para o modelo é a seleção da leitura apropriada de um verbo a partir do contexto. Isto é conseguido quase da mesma forma que a resolução da ambigüidade de substantivos. As leituras diferentes são representadas por (potencialmente) conjuntos diferentes de microcaracterísticas semânticas; por exemplo, a leitura Agente/Não-Instrumento de *quebrou* (quebrouAVP) envolve contato entre o Agente e o Paciente, enquanto que a versão Instrumento/Não-Agente (quebrouIVP) e a versão Agente/Instrumento (quebrouAVPI) envolve contato entre o Instrumento e o Paciente. A representação de entrada das características de um dado verbo é a mesma, não importando o contexto, e a tarefa dada ao modelo é ativar o conjunto de características para a versão apropriada da sentença. Ao invés de usar o padrão médio baseado em todas as diferentes leituras possíveis do verbo, usou-se um padrão "genérico" para cada verbo, que é o padrão mais típico do "frame" de caso do verbo. Isto é indicado na figura 3.2 pelo padrão de características próximo ao verbo regular.

Os padrões de características correspondentes aos diferentes "frames" de caso entre os quais o modelo deve escolher, são indicados nas linhas da figura seguindo seu padrão genérico. (Os rótulos nestas linhas são usados simplesmente para designar os padrões de características. Eles indicam os casos que os vários argumentos na estrutura de superfície da sentença têm. Portanto, quebrouAVPI especifica o "frame" de caso no qual o sujeito de superfície é o Agente, o objeto de superfície é o Paciente e o com-SN é o Instrumento.) Note que as microcaracterísticas de duas leituras diferentes do mesmo verbo podem ou não diferir, dependendo se as características do cenário mudam ou não em "frames" de casos diferentes.

Os vetores de características para os constituintes da sentença *O garoto quebrou a vidraça com o martelo* são mostrados no alto das figuras 3.3. Note que as estruturas de sentença estão mostradas na ordem: Verbo, SN Sujeito, SN Objeto e Com-SN. A linha de letras abaixo dos vetores de características indica a primeira letra do nome da dimensão na qual cada característica representa um valor. Por exemplo, os primeiros dois elementos do vetor de características do verbo são rotulados r para a dimensão REALIZADOR; os primeiros dois valores para cada vetor de características de substantivos são rotulados h para a dimensão HUMANO.

- Unidades de estrutura de sentença

A representação do nível de estrutura de sentença de uma sentença de entrada não é o conjunto de vetores de características constituintes, mas sim o padrão de ativação que esses

vetores produzem sobre as unidades que correspondem a *pares* de características. Estas unidades são chamadas unidades de estrutura de sentença (ES).

Cada unidade ES representa a conjunção de duas microcaracterísticas do preenchedor de um caso de superfície particular. Como há quatro casos de estrutura de sentença, existem quatro conjuntos de unidades ES. Dentro de cada conjunto existe uma unidade que representa a conjunção de todo valor de microcaracterística em cada dimensão com todo valor de microcaracterística em qualquer outra dimensão. Por exemplo, para os substantivos, existem unidades para:

HUMANO = sim / GÊNERO = masculino
SUAVIDADE = duro / DUREZA = frágil

entre muitas outras; para o verbo, uma das unidades corresponde a

REALIZADOR = sim / TOQUE = instrumento

(isto é, existe um realizador - o Instrumento toca o Paciente).

As unidades de estrutura de sentença são mostradas na figura 3.3 em quatro matrizes triangulares (figuras 3.3A, 3.3B, 3.3C e 3.3D). A matriz do verbo é diferente das matrizes dos três SNs pois características diferentes são juntadas nas representações de verbo e de SN.

Cada matriz contém as unidades conjuntivas para o constituinte imediatamente acima dela. Existe uma unidade onde quer que exista um 1, um "?" ou um ".". Dentro de cada matriz, as unidades são colocadas de tal forma que a coluna em que a unidade está indica uma das microcaracterísticas que ela representa, e a linha em que ela está, indica a outra microcaracterística. Linhas e colunas são ordenadas da mesma forma que os vetores de microcaracterísticas no alto da figura. As dimensões são indicadas pela linha de letras ao longo do alto de cada matriz e ao longo da esquerda. Note que o conjunto de unidades em cada matriz preenche menos da metade de cada bloco por duas razões. Primeiro, existem apenas [n(n-1)]/2 pares distintos de características n; segundo, pares de valores na mesma dimensão não estão incluídos.

```
  1.1..1..1....1...1.1
  rrccttttnnnnaaapppii
r
r
c 1?
c ?.
t ?.?.
t 1?1?
t ?.?.
t ?.?.
n 1?1??1??
n ?.?..?..
n ?.?..?..
n ?.?..?..
a ?.?..?..?...
a 1?1??1??1???
a ?.?..?..?...
p ?.?..?..?....?.
p ?.?..?..?....?.
p 1?1??1??1????1?
i ?.?..?..?....?...?
i 1?1??1??1????1???1
```

Figura 3.3A. Estrutura de sentença para "quebrou".

```
  1.1.1..1...1.1.1...1
  hhssggvvvfffppddtttt
h
h
s 1?
s ?.
g 1?1?
g ?.?.
v ?.?.?.
v 1?1?1?
v ?.?.?.
f .?.?..?.
f ?.?.?..?.
f 1?1?1??1?
p ?.?.?..?...?
p 1?1?1??1???1
d ?.?.?..?...?.?
d 1?1?1??1???1?1
t ?.?.?..?...?.?.?
t ?.?.?..?...?.?.?
t 1?1?1??1???1?1?1
t ?.?.?..?...?.?.?
```

Figura 3.3B. Estrutura de sentença para "garoto".

```
  .1.1.1.1..1.1.1...1.
  hhssggvvvfffppddtttt
h
h
s .?
s ?1
g .?.?
g ?1?1
v .?.?.?
v ?1?1?1
v .?.?.?
f .?.?.?.?.
f ?1?1?1?1?
f .?.?.?.?.
p ?1?1?1?1??1?
p .?.?.?.?..?.
d ?1?1?1?1??1?1?
d .?.?.?.?..?.?.
t .?.?.?.?..?.?.?.
t .?.?.?.?..?.?.?.
t ?1?1?1?1??1?1?1?
t .?.?.?.?..?.?.?.
```

Figura 3.3C. Estrutura de sentença para "vidraça".

```
  .1.11.1..1...1.1..1.
  hhssggvvvfffppddtttt
h
h
s .?
s ?1
g ?1?1
g .?.?
v ?1?11?
v .?.??.
v .?.??.
f .?.??.?..
f ?1?11?1??
f .?.??.?..
p .?.??.?...?.
p ?1?11?1???1?
d .?.??.?...?..?
d ?1?11?1???1??1
t .?.??.?...?..?.?
t .?.??.?...?..?.?
t ?1?11?1???1??1?1
t .?.??.?...?..?.?
```

Figura 3.3D. Estrutura de sentença para "martelo".

Considera-se vários esquemas para a ativação das unidades de estrutura de sentença. Um esquema possível seria usar uma regra de ativação determinística, tal que uma unidade ES particular seria ligada apenas se ambas as características que a unidade representa estivessem dentro do vetor de características. Este uso das unidades ES permitiria que o modelo aprendesse a responder de uma forma mais apurada a determinadas conjunções de microcaracterísticas. Entretanto, deseja-se ver o quão bem o modelo poderia funcionar usando uma representação de entrada com ruídos. Contudo sabe-se que a generalização é facilitada quando as unidades que casam apenas parcialmente com a entrada têm alguma chance de serem ativadas. No presente caso, considera-se isto importante para ser capaz de generalizar para palavras com significados similares. Portanto, as unidades ES são tratadas como unidades binárias estocásticas. Cada unidade ES recebe entrada excitatória de cada uma das duas características que ela representa e a polarização e a variância das unidades é colocada de tal forma que quando ambas as características das unidades ES estão ativas, a unidade "liga" com probabilidade .85; e quando nenhuma está ativa, ela "liga" com probabilidade .15. Estes casos são representados na tabela por "1" e ".", respectivamente. As unidades que recebem uma entrada excitatória "liga" com probabilidade .5; estas unidades são representadas por "?".

Enquanto o modelo funciona bem com esta simulação, presume-se que simulações que usem um léxico maior requereria maior diferenciação de algumas representações de substantivos e verbos. Para trabalhar com tais casos, acredita-se que seria necessário permitir o ajuste das conexões de entrada às unidades ES através do algorítmo "backpropagation" de tal forma que uma diferenciação maior possa ser obtida quando necessário.

3.3.2.4. Representação de caso

A representação de caso tem uma forma levemente diferente da representação de estrutura de sentença. Para entender esta representação, é útil voltar a um ponto de vista mais abstrato e considerar mais geralmente como se deve representar uma descrição estrutural numa representação distribuída. Em geral uma descrição estrutural pode ser representada por um conjunto de triplas da forma (A R B) onde A e B correspondem aos nós na descrição estrutural e R representa a relação entre os nós. Por exemplo, uma hierarquia de inclusão de classes pode ser representada por triplas da forma (X É-UM Y), onde X e Y são nomes de categorias. Qualquer outra descrição estrutural, seja uma estrutura de constituinte sintático, uma estrutura de constituinte semântico ou qualquer outra coisa, pode ser representada desta forma. Especificamente, a atribuição

de caso dos constituintes da sentença *O garoto quebrou a vidraça com o martelo* pode ser representada como:

Quebrou Agente Garoto
Quebrou Paciente Vidraça
Quebrou Instrumento Martelo

A estrutura constituinte de uma sentença tal como *O garoto comeu o macarrão com molho* poderia ser representada por:

Comeu Agente Garoto
Comeu Paciente Macarrão
Macarrão Modificador Molho

Numa representação local, poder-se-ia representar cada uma destas triplas por uma unidade única. Cada unidade destas representaria então a conjunção de uma determinada cabeça ou lado esquerdo de uma tripla, uma determinada relação, e uma determinada cauda ou lado direito. Nesta abordagem mais distribuída seriam alocados grupos de unidades para representar cada uma das possíveis relações (ou casos), que seriam o Agente, Paciente, Instrumento e Modificador, e teriam unidades dentro de cada grupo representando conjunções de microcaracterísticas do primeiro e terceiro argumentos (a cabeça e a cauda) da tripla. Portanto, a tripla é representada não por uma única unidade ativa, mas por um padrão de ativação sobre um conjunto de unidades.

Nesta implementação, existe um grupo de unidades para cada uma das quatro relações permitidas na estrutura de caso. Os grupos do Agente, Paciente, Instrumento e Modificador são colocados da esquerda para a direita. Dentro de cada grupo, as unidades individuais representam conjunções de uma microcaracterística da cabeça de cada relação com uma microcaracterística da cauda de cada relação. Então, por exemplo na figura 3.4A, *Quebrou-Agente-Garoto* é representado por um padrão de ativação. A unidade na i-ésima linha e j-ésima coluna representa a conjunção da característica i do verbo com característica j do substantivo. Então, todas as unidades com a mesma característica de verbo estão alinhadas na mesma linha, enquanto que todas as unidades com a mesma característica de substantivo estão alinhadas na mesma coluna. Para o grupo do Modificador, a unidade na i-ésima linha e j-ésima coluna representaria a conjunção da característica i do SN modificado e a característica j do SN modificador. As letras indicando as especificações de dimensão das unidades são colocadas ao longo das bordas lateral e superior.

3.3.2.5. Detalhes do processamento de sentença e aprendizado

Na apresentação de uma sentença, a entrada da rede para cada uma das unidades de estrutura da sentença é determinada, baseada nos vetores de características das palavras. Cada uma dessas unidades é então ligada probabilisticamente. Cada unidade de estrutura de superfície tem uma conexão modificável para cada uma das unidades de estrutura de caso. Em adição, cada unidade de estrutura de caso tem uma polarização modificável (equivalente a uma conexão a partir de uma unidade especial que está sempre ligada). Baseado no padrão de estrutura de sentença e nos valores correntes dos pesos, uma entrada da rede para cada unidade de estrutura de caso é computada. Unidades de estrutura de caso têm valores de ativação 0 e 1 e a ativação é uma função probabilística da entrada da rede, que é implementada com o algoritmo "backpropagation".

Durante o aprendizado, a ativação resultante de cada unidade de estrutura de caso é comparada ao valor que ela deveria ter na leitura correta da sentença (figura 3.4). A leitura correta é fornecida como uma "entrada mestre" especificando quais unidades de casos devem estar ligadas. A idéia é que esta entrada mestre seja análoga a representação que um aprendiz de linguagem real deveria construir da situação na qual a sentença ocorreria. O aprendizado simplesmente corresponde ao ajuste de pesos de conexão para fazer a saída gerada pelo modelo corresponder o mais próximo possível à entrada mestre.

3.3.2.6. Experimentos de simulação

A coisa mais importante sobre o modelo é o fato de que sua resposta a novos impulsos é estritamente dependente de sua experiência. Na avaliação de seu comportamento então, é importante ter conhecimento ao que ele foi exposto durante o aprendizado.

```
  hhssggvvvfffppddtttt
r 1.1.1..1...1.1.1..1.
r ....................
c 1.1.1..1...1.1.1..1.
c ....................
t ....................
t 1.1.1..1...1.1.1..1.
t ....................
t ....................
n 1.1.1..1...1.1.1..1.
n ....................
n ....................
n ....................
a ....................
a 1.1.1..1...1.1.1..1.
a ....................
p ....................
p ....................
p 1.1.1..1...1.1.1..1.
i ....................
i 1.1.1..1...1.1.1..1.
```

Figura 3.4A. Estrutura de caso para "garoto-quebrou".

```
  hhssggvvvfffppddtttt
r .1.1.1.1..1.1.1...1.
r ....................
c .1.1.1.1..1.1.1...1.
c ....................
t ....................
t .1.1.1.1..1.1.1...1.
t ....................
t ....................
n .1.1.1.1..1.1.1...1.
n ....................
n ....................
n ....................
a ....................
a .1.1.1.1..1.1.1...1.
a ....................
p ....................
p ....................
p .1.1.1.1..1.1.1...1.
i ....................
i .1.1.1.1..1.1.1...1.
```

Figura 3.4B. Estrutura de caso para "quebrou-vidraça".

```
  hhssggvvvfffppddtttt
r .1.11.1...1..1.1..1.
r ....................
c .1.11.1...1..1.1..1.
c ....................
t ....................
t .1.11.1...1..1.1..1.
t ....................
t ....................
n .1.11.1...1..1.1..1.
n ....................
n ....................
n ....................
a ....................
a .1.11.1...1..1.1..1.
a ....................
p ....................
p ....................
p .1.11.1...1..1.1..1.
i ....................
i .1.11.1...1..1.1..1.
```

Figura 3.4C. Estrutura de caso para "quebrou-martelo".

A experiência principal consiste na geração de várias sentenças derivadas dos "frames" de sentenças listados na tabela 3.2 (McClelland e Kawamoto, 1986). Deve ser enfatizado que estes "frames" de sentenças são simplesmente usados para gerar um conjunto de sentenças válidas. Cada "frame" especifica um verbo, um conjunto de casos e uma lista de possíveis preenchedores de cada caso. Portanto, o "frame" de sentença *O humano quebrou o objeto_frágil com o quebrador* é simplesmente um gerador para todas as sentenças na qual *humano* é substituído por uma das palavras na lista de humanos na tabela 3.3 (McClelland e Kawamoto, 1986), *objeto_frágil* é substituído por uma das palavras na lista de objetos frágeis na tabela 3.3 e *quebrador* é substituído por uma das palavras da lista de quebradores na tabela. É claro que estes geradores não capturam todas as propriedades dos elementos envolvidos em cenários reais e então não se pode esperar que o modelo represente fielmente todas as sutilezas. Entretanto, existem certos fatos implícitos na montagem completa das sentenças pelos geradores. Por exemplo, todos os quebradores exceto um são *duros* (apenas *bola* é codificada como *mole* nos padrões das características); apenas os *humanos* entram como Agentes em cenários envolvendo o uso de Instrumento; etc.

As representações de "frame" de caso "alvo" de sentenças são geradas através das próprias sentenças. As atribuições de caso são indicadas na tabela 3.2 pela seqüência de letras maiúsculas. Estas indicam a atribuição de argumentos a partir das sentenças para os casos do Agente, Verbo, Paciente, Instrumento e Modificador (do Paciente)[13]. Note que existem algumas sentenças que poderiam ser geradas por mais de um gerador. Portanto, *O garoto bateu na menina com a bola* pode ser gerada pelo gerador *O humano bateu no humano com a propriedade*, no qual *a bola* é tratada como um Modificador do Paciente. Alternativamente, ela poderia ser gerada pelo gerador *O humano bateu na coisa com o batedor*. Neste caso, *a bola* é tratada como o Instrumento. Similarmente, *O macaco quebrou o vaso* pode ser gerada por *O quebrador quebrou o objeto_frágil*, cujo caso sua representação de "frame" de caso contém um equipamento para trocar pneus servindo como Instrumento. A mesma sentença pode também ser gerada por *O animal quebrou o objeto_frágil*, em cujo caso, é claro, sua representação de "frame" de caso contém um animal servindo como Agente.

[13] Dois casos especiais devem ser notados: para "os humanos comiam", o frame de caso contém uma especificação (F) que designa uma paciente implícito que é o alimento genérico com forma não especificada, como indicado nos padrões de características mostrados na figura 3.1. Para "o humano moveu-se" e "o animal moveu-se", o frame de caso contém uma especificação (S) que indica que existe um paciente implícito que é o mesmo do agente (*mover* é reflexivo).

TABELA 3.2

GERADORES PARA SENTENÇAS USADAS NO TREINAMENTO E TESTES

"FRAMES" DE SENTENÇA	ATRIBUIÇÃO DE ARGUMENTOS
O humano comeu.	AVF
O humano comeu o alimento.	AVP
O humano comeu o alimento com o alimento.	AVPM
O humano comeu o alimento com o utensílio.	AVPI
O animal comeu.	AVF
O predador comeu a presa.	AVP
O humano quebrou o objeto_frágil.	AVP
O humano quebrou o objeto_frágil com o quebrador.	AVPI
O quebrador quebrou o objeto_frágil.	IVP
O animal quebrou o objeto_frágil.	AVP
O objeto_frágil quebrou.	PV
O humano bateu na coisa.	AVP
O humano bateu no humano com a propriedade.	AVPM
O humano bateu na coisa com o batedor.	AVPI
O batedor bateu na coisa.	IVP
O humano moveu-se.	AVS
O humano moveu o objeto.	AVP
O animal moveu-se.	AVS

Nota: Atribuições de argumentos especificam a atribuição de caso dos constituintes de uma sentença da esquerda para a direita. A = Agente, V = Verbo, P = Paciente, I = Instrumento, M = Modificador, F = Alimento (implícito), S = Reflexivo ("Self"). (McClelland e Kawamoto, 1986)

TABELA 3.3
CATEGORIAS DE SUBSTANTIVOS

humano	homem mulher garoto menina
animal	macaco-si galinha-vi cachorro lobo ovelha leão
objeto	bola macaco-me queijo galinha-co cortina escrivaninha boneca garfo martelo prato pedra macarrão colher cenoura vaso vidraça
coisa	humano animal objeto
predador	lobo leão
presa	galinha-vi ovelha
alimento	galinha-co queijo cenoura
utensílio	garfo colher
objeto_frágil	prato vidraça vaso
batedor	bola macaco-me martelo pedra vaso
quebrador	bola macaco-me martelo pedra
propriedade	bola cachorro macaco-me boneca martelo vaso

Nota: macaco-si = macaco símio (animal); macaco-me = macaco mecânico (instrumento para trocar pneus) e galinha-vi = galinha viva (animal); galinha-co = galinha cozida (alimento).

Para o experimento principal, a tabela 3.3 foi implementada apenas com duas palavras de cada classe para alimentar o gerador automático de sentenças. Isto para permitir que o treinamento da rede dê-se apenas com algumas sentenças possíveis, as sentenças de testes **familiares**. Mas foram implementadas todas as palavras da figura 3.1, ou seja, todas as outras sentenças possíveis, previstas pelas tabelas 3.2 e 3.3 são consideradas sentenças **novas**. Estas sentenças não foram usadas para treinar o modelo[14].

Ao modelo foi dado 20 ciclos de treinamento, com o conjunto de sentenças de treino. Em cada ciclo, cada sentença era apresentada, a resposta do modelo era gerada e os pesos de conexão era ajustados de acordo com o procedimento "backpropagation".

[14] Alguns geradores (por exemplo, *o humano bateu na coisa com o batedor*) geram um grande número de sentenças diferentes, mas outros (por exemplo, *o humano comeu, o predador comeu a presa*) geram poucas sentenças. Por esta razão, limitou-se em duas palavras para cada tipo de frase (tabela 3.3).

3.3.3. Redes "backpropagation"

A habilidade para treinar redes com várias camadas é um passo importante na direção da construção de máquinas inteligentes a partir de componentes parecidos com os neurônios. A meta é pegar uma massa de elementos processadores, que simulam a célula nervosa, e ensiná-la a realizar tarefas úteis. É desejável que ela seja rápida e resistente a danos. É desejável que generalize a partir das entradas que vê.

O que uma rede multicamadas pode calcular? A resposta é: qualquer coisa. Dado um conjunto de entradas, pode-se usar unidades de limiar como simples portas AND, OR e NOT, arranjando apropriadamente o limiar e os pesos de conexão. Sabe-se que é possível construir qualquer circuito combinacional a partir destas unidades lógicas básicas.

O maior problema é o aprendizado. O sistema de representação de conhecimento empregado pelas redes neurais é um tanto obscuro: as redes devem aprender suas próprias representações porque programá-las a mão é impossível. Uma propriedade das redes neurais diz que tudo que elas podem calcular, elas podem aprender a calcular.

É útil tratar primeiro com uma subclasse de redes multicamadas, chamadas de redes **totalmente conectadas**, divididas em **camadas** e alimentadas **para frente**. Um exemplo de tal rede é mostrada na figura 3.5. Nesta figura x_i, h_i e o_i representam os níveis de unidade de ativação das unidades de entrada, escondida e saída, respectivamente. Os pesos das conexões entre as camadas de entrada e escondida são denotados por $w1_{ij}$, enquanto que os pesos das conexões entre as camadas escondida e de saída são denotados por $w2_{ij}$. Esta rede tem três camadas, ainda que seja possível, e algumas vezes útil, ter mais de três. Cada unidade numa camada é conectada a toda unidade da próxima camada na direção para frente, ou seja, cada unidade da camada de entrada é conectada a todas as unidades da camada escondida, nesta direção. As ativações fluem a partir da camada de entrada através da camada escondida, para a camada de saída. O conhecimento da rede é codificado nos pesos das conexões entre as unidades. Os níveis de ativação das unidades da camada de saída determinam a saída da rede.

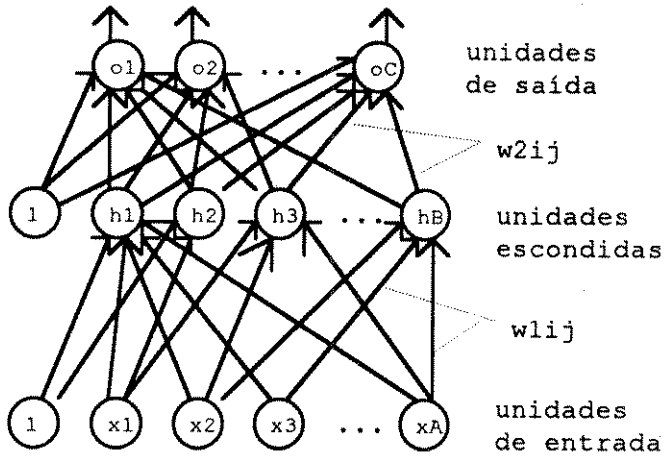


Figura 3.5. Uma rede multicamada (Rich e Knight, 1991).

A existência da camada escondida permite que a rede desenvolva representações internas. O comportamento destas unidades escondidas é automaticamente aprendido, não é pré-programado.

A maior propriedade dos sistemas conexionistas é que a rede neural não aprende apenas a classificar as entradas nas quais ela é treinada, mas também a **generalizar** e ser capaz de classificar entradas nunca vistas.

Tudo que as redes neurais parecem capazes de fazer é classificar. Os graves problemas da Inteligência Artificial, como planejamento, análise de linguagem natural e prova de teorema, não são simplesmente tarefas de classificação, então como as redes neurais resolvem estes problemas? Resolver os problemas de classificação são, no presente, o que as redes neurais fazem melhor. Mas pesquisa-se a aplicação aos outros problemas, como processamento de linguagem natural, que é o tema deste trabalho.

Uma limitação das redes atuais é como elas tratam com fenômenos que envolvem o tempo. Esta limitação é resolvida, de certa forma, pelas redes recorrentes, mas os problemas ainda são muitos.

A unidade na rede "backpropagation" requer uma função de ativação baseada numa sigmóide (ou forma de *S*) que é contínua e diferenciável. Uma unidade soma suas entradas pondera-

das e produz como saída um valor real entre 0 e 1. Seja *soma* a soma ponderada das entradas de uma unidade. A equação para a saída da unidade é dada por:

$$\text{saída} = \frac{1}{1 + e^{-\text{soma}}}$$

Uma rede "backpropagation" tipicamente inicia com um conjunto de pesos aleatórios. A rede ajusta seus pesos cada vez que ela vê um par entrada-saída. Cada par requer dois estágios: um passo para frente e um passo para trás. O passo para frente envolve a apresentação de uma amostra de entrada à rede e as ativações propagam-se até alcançarem a camada de saída. Durante o passo para trás, a saída real da rede (do passo para frente) é comparada com a saída desejada e as estimativas de erro são calculadas para as unidades de saída. Os pesos conectados às unidades de saída podem ser ajustados a fim de reduzir estes erros. Pode-se usar as estimativas de erro das unidades de saída para derivar as estimativas de erro para as unidades das camadas escondidas. Finalmente, os erros são propagados de volta às conexões que tiveram origem nas unidades de entrada.

O algorítmo "backpropagation" geralmente atualiza seus pesos depois de ver cada par entrada-saída. Depois de visto todos os pares entrada-saída (e ajustados seus pesos muitas vezes), diz-se que uma **época** completou-se. O treinamento de rede "backpropagation" usualmente requer muitas épocas.

3.3.3.1. O algorítmo "backpropagation"

O algorítmo seguinte é baseado na estrutura básica da figura 3.5 (Rich e Knight, 1991).

Algorítmo "Backpropagation"

Dado: Um conjunto de pares de vetores de entrada-saída.

Calcular: Um conjunto de pesos para uma rede de três camadas que mapeia entradas nas saídas correspondentes.

1. Seja A o número de unidades na camada de entrada, como determinado pelo comprimento dos vetores de treinamento de entrada. Seja C o número de unidades na camada de saída. Agora escolher B, o número de unidades na camada escondida[15]. Como mostrado na figura 3.5, as camadas de entrada e escondida têm uma unidade extra usada para limiar; portanto, as unidades nestas camadas serão indexadas pela faixa (0,...,A) e (0,...,B). Denota-se os níveis de ativação das unidades na camada de entrada por x_j, na camada escondida por h_j e na camada de saída por o_j. Os pesos conectando a camada de entrada à camada escondida são denotados por $w1_{ij}$, onde o índice i indexa as unidades de entrada e o índice j indexa as unidades escondidas. Da mesma forma, os pesos conectando a camada escondida à camada de saída são denotados por $w2_{ij}$, com i indexando as unidades escondidas e j indexando as unidades de saída.

2. Iniciar os pesos da rede. A cada peso deve ser atribuído um valor aleatório entre -0.1 e 0.1.

$$w1_{ij} = \text{random}(-0.1,0.1) \quad \text{para todo} \quad i = 0,..A, j = 1,...,B$$
$$w2_{ij} = \text{random}(-0.1,0.1) \quad \text{para todo} \quad i = 0,..,B, j = 1,...,C$$

3. Iniciar as ativações para as unidades de limiar. Os valores destas unidades nunca devem mudar.

$$x_0 = 1.0$$
$$h_0 = 1.0$$

4. Escolher um par entrada-saída. Suponha que o vetor de entrada seja x_i, e o vetor de saída desejada seja y_i. Atribuir níveis de ativação às unidades de entrada.

---

[15] Para o analisador semântico, escolheu-se B = 39, na relação 174-39-400. Para o analisador recorrente, escolheu-se B = 80, para a relação 20-80-20. Estes valores foram obtidos experimentalmente, considerando as limitações da máquina.

5. Propagar as ativações a partir das unidades na camada de entrada para as unidades na camada escondida usando a função de ativação sigmóide:

$$h_j = \frac{1}{1 + e^{-soma}} \quad \text{para todo } j = 1,...,B$$

onde $soma = \Sigma^{A}_{i=0} w1_{ij}x_i$. Note que i varia de 0 a A. $w1_{0j}$ é o peso do limiar para a unidade escondida j (sua propensão a "disparar"[16], a despeito de suas entradas). x_0 é sempre 1.0.

6. Propagar as ativações a partir das unidades na camada escondida para as unidades na camada de saída.

$$o_j = \frac{1}{1 + e^{-soma}} \quad \text{para todo } j = 1,...,C$$

onde $soma = \Sigma^{B}_{i=0} w2_{ij}h_i$. Novamente, o peso de limiar $w2_{0j}$ para a unidade de saída j traz uma contribuição à soma ponderada. h_0 é sempre 1.0.

7. Calcular os erros[17] das unidades na camada de saída, denotado por $\delta 2_j$. Os erros são baseados na saída real da rede (o_j) e na saída desejada (y_j).

$$\delta 2_j = o_j(1 - o_j)(y_j - o_j) \quad \text{para todo} \quad j = 1,...,C$$

8. Calcular os erros das unidades na camada escondida, denotado por $\delta 1_j$.

$$\delta 1_j = h_j(1 - h_j) \Sigma^{C}_{i=1} \delta 2_i \cdot w2_{ji} \quad \text{para todo} \quad j = 1,...,B$$

---

[16] Disparar é tornar-se igual a 1.0.

[17] A fórmula do erro é relacionada à derivada da função de ativação.

9. Ajustar os pesos entre a camada escondida e a camada de saída. A taxa de aprendizado é denotada por η. Um valor razoável para η é 0.35[18].

$$\Delta w2_{ij} = \eta \cdot \delta 2_j \cdot h_i \quad \text{para todo} \quad i = 0,...B, j = 1,...,C$$

10. Ajustar os pesos entre a camada de entrada e a camada escondida.

$$\Delta w1_{ij} = \eta \cdot \delta 1_j \cdot x_i \quad \text{para todo} \quad i = 0,...A, j = 1,...,B$$

11. Ir para o passo 4 e repetir. Quando todas os pares de entrada-saída estiverem sido apresentados à rede, uma época completou-se. Repetir os passos 4 a 10 para quantas épocas desejar.

O algorítmo pode ser generalizado para redes com mais de três camadas[19]. A velocidade do aprendizado pode ser aumentada alterando os passos de modificação de pesos 9 e 10, com a inclusão de um termo α. As fórmulas de atualização de pesos ficam:

$$\Delta w2_{ij}(t+1) = \eta \cdot \delta 2_j \cdot h_i + \alpha \Delta w2_{ij}(t)$$

$$\Delta w1_{ij}(t+1) = \eta \cdot \delta 1_j \cdot x_i + \alpha \Delta w1_{ij}(t)$$

onde h_i, x_i, $\delta 1_j$ e $\delta 2_j$ são medidos no tempo t + 1. $\Delta w_{ij}(t)$ é a mudança que o peso experimenta durante o passo para frente-para trás anterior. Se α é colocado em 0.9, a velocidade de aprendizado aumenta[20].

---

[18] Usou-se, para o analisador semântico, η = 0.35. Já para a rede recorrente, usou-se η = 0.25, valor escolhido experimentalmente.

[19] Uma rede com três camadas (uma única camada escondida) pode calcular qualquer função que uma rede com muitas camadas escondidas pode calcular. Entretanto, o aprendizado é, às vezes, mais rápido, com múltiplas camadas escondidas (Rich e Knight). Mas, para este trabalho, a melhor solução foi mesmo uma rede com três camadas.

[20] Empiricamente, os melhores resultados acontecem quando α é zero para os primeiros passos de treinamento, aumentando seu valor gradativamente até 0.9 durante o treinamento, segundo Rich e Knight. Para o analisador semântico, cuja fase de aprendizado consiste de 20 épocas, manteve-se α = 0.0 para as cinco primeiras épocas, aumentando seu valor de 0.3 em 0.3 para as épocas seguintes, até alcançar 0.9.

Como a função de ativação tem a forma sigmóide, pesos infinitos seriam necessários para as saídas reais da rede alcançarem 0.0 e 1.0, portanto, as saídas desejadas (os $y_j's$ dos passos 4 e 7 acima) são usualmente dadas como 0.1 e 0.9. A sigmóide é útil para a rede "backpropagation", pois a derivação da regra de atualização do peso requer que a função de ativação seja contínua e diferenciável.

3.3.3.2. Generalização

Se todas as entradas e saídas possíveis são mostradas à uma rede "backpropagation", a rede achará (provavelmente, eventualmente) um conjunto de pesos que mapeia as entradas nas saídas. Para muitos problemas de Inteligência Artificial, entretanto, é impossível fornecer todas as entradas possíveis. Para resolver este problema, a rede "backpropagation" é boa no mecanismo de generalização. Se se trabalha num domínio onde entradas similares são mapeadas em saídas similares[21], a rede "backpropagation" irá interpolar quando forem fornecidas entradas que a rede nunca viu antes.

3.3.4. Redes Recorrentes

Uma deficiência clara nos modelos de redes neurais comparados aos modelos simbólicos é a dificuldade que eles têm em tratar com tarefas temporais em Inteligência Artificial tais como planejamento e análise de linguagem natural. As redes recorrentes, ou redes com loops, são uma tentativa de corrigir esta situação.

Considere a tentativa de ensinar uma rede como arremessar uma bola de basquete à cesta (Rich e Knight). Pode-se apresentar a rede como uma situação de entrada (distância e altura da cesta, posição inicial dos músculos), mas necessita-se mais que um simples vetor de saída. Necessita-se de uma série de vetores de saída: primeiro mova os músculos desta forma, depois desta forma, etc. A "rede de Jordan" faz algo parecido com isto. É mostrada na figura 3.6. As **unidades de plano** da rede permanecem constantes. Elas correspondem a uma instrução como "arremessar uma bola a cesta". As **unidades de estado** codificam o estado corrente da rede. As

[21] No caso deste trabalho, como as palavras são descritas por microcaracterísticas semânticas, existem palavras próximas no significado (como, por exemplo, *homem* e *menino*), que têm muitas microcaracterísticas em comum, ou seja, seus vetores de microcaracterísticas são próximos.

unidades de saída simultaneamente dá comandos (por exemplo, movimente o braço **x** para a posição **y**) e atualiza as unidades de estado. A rede nunca se estabiliza, ou seja, nunca alcança um estado estável; ao invés disto, ela muda a cada passo de tempo.

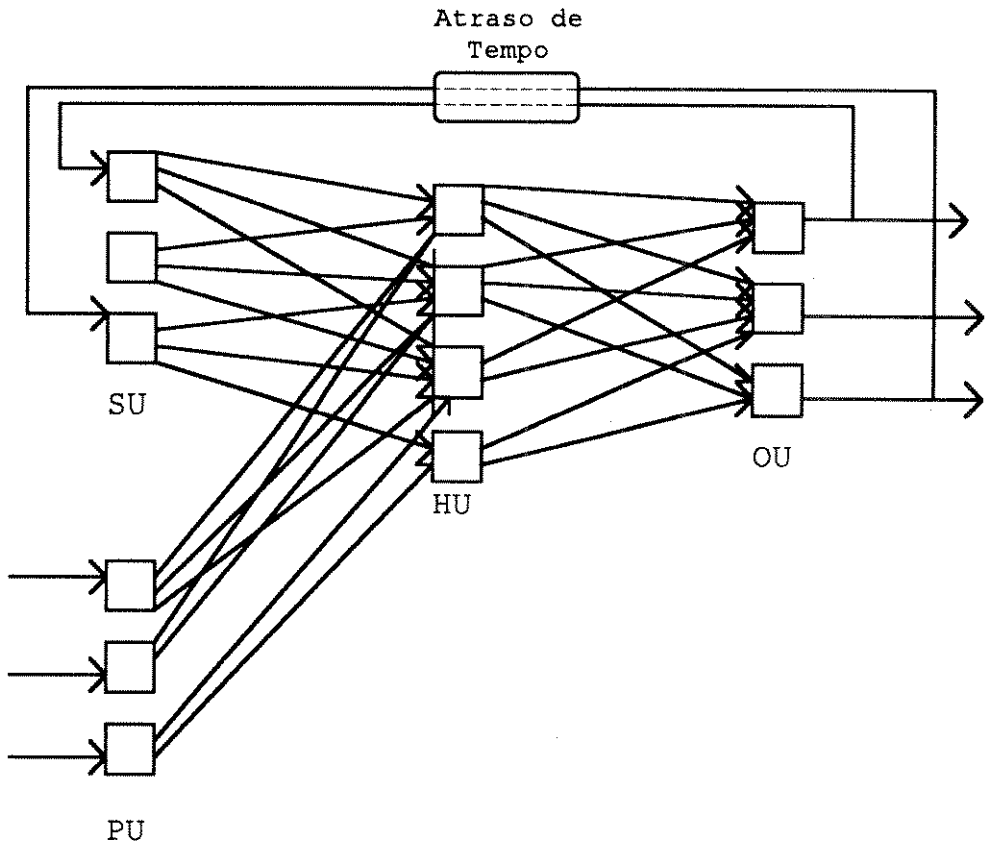


Figura 3.6. Uma rede de Jordan, onde SU = unidades de estado, PU = unidades de plano, HU = unidades escondidas e OU = unidades de saída.

As redes recorrentes podem ser treinadas com o algorítmo "backpropagation". A cada passo, compara-se as ativações das unidades de saída com as ativações desejadas e os erros são propagados de volta através da rede. Quando o treinamento está completo, a rede ainda é capaz de realizar uma seqüência de ações. Características de "backpropagation", tal como a generalização automática, também ocorrem nas redes recorrentes. Entretanto, poucas modificações são úteis. Primeiro, deseja-se que as unidades de estados mudem suavemente. A suavidade pode ser implementada como uma mudança na regra de atualização de peso; essencialmente, o "erro" de uma saída torna-se uma combinação do erro real e da magnitude da mudança nas

unidades de estado. O reforço da restrição da suavidade torna-se muito importante no aprendizado rápido, já que ele remove muitas das opções de manipulação de peso disponíveis no "backpropagation".

Um problema maior nos sistemas de aprendizado supervisionado ocorre na correção do comportamento da rede. Se dados de treinamento suficientes podem ser coletados, então saídas alvo podem ser providas para muitos vetores de entrada. Entretanto, as redes recorrentes têm problemas de treinamento especiais, por causa da dificuldade de especificar completamente uma série de saídas alvo. No arremesso de bolas de basquete, por exemplo, a retroalimentação vem do mundo externo (isto é, onde a bola cai), não de um professor mostrando como mover cada músculo. Para contornar esta dificuldade, pode-se aprender um **modelo mental**, um mapeamento que relaciona as saídas da rede aos eventos no mundo. Com tal modelo, uma vez conhecido, o sistema proposto pode aprender tarefas seqüenciais pela propagação de volta ("backpropagation") dos erros que ele vê no mundo real. Então isto é necessário para aprender duas coisas diferentes: o relacionamento entre o plano e a saída da rede e entre a saída da rede e o mundo real.

As redes deste tipo são essencialmente iguais a da figura 3.6, exceto pela adição de mais duas camadas: uma outra camada escondida e uma camada representando os resultados como visto no mundo. Primeiro, a última porção mencionada da rede é treinada (usando "backpropagation") em vários pares de saídas e alvos até que a rede consiga saber como suas saídas afetam o mundo real. Depois disto os pesos brutos são estabelecidos, a rede inteira é treinada usando retroalimentação do mundo real até que ela seja capaz de funcionar bem.

Um outro tipo de rede recorrente é descrita por Elman (1990). Neste modelo, os níveis de ativação são explicitamente copiados das unidades escondidas para as unidades de estado. As redes deste tipo têm sido usadas em várias aplicações, incluindo análise de linguagem natural. O trabalho proposto foi baseado no trabalho de Elman, porém os níveis de ativação são copiados das unidades de saída para as unidades de estado.

3.3.4.1. A Representação do tempo

A questão de como representar o tempo em modelos conexionistas é muito importante. Uma abordagem é representar o tempo implicitamente pelos seus efeitos no processamento ao invés de explicitamente (como numa representação espacial).

O tempo é muito importante em cognição. Está intrinsicamente ligado a muitos comportamentos (como na linguagem) que se expressam como seqüências temporais. A questão de como representar o tempo parece ser um problema unicamente dos modelos de processamento paralelo, mas mesmo em sistemas tradicionais (seriais) a representação da ordem serial e a interação de uma entrada serial ou saída com níveis mais altos de representação apresenta desafios. Os lingüistas teóricos têm tentado se preocupar menos com a representação de processamento de aspectos temporais de discursos (assumindo, por exemplo, que toda a informação num discurso é tida simultaneamente numa árvore sintática); mas a pesquisa na análise de linguagem natural sugere que o problema não é trivialmente resolvido. Portanto, o que é um dos mais elementares fatos sobre a atividade humana - que tem extensão temporal - é algumas vezes ignorado e é freqüentemente problemático.

Nos modelos de processamento paralelo distribuído, o processamento de entradas seqüenciais é completado de muitas formas. A solução mais comum é "paralelizar o tempo", dando a ele uma representação espacial. Entretanto, existem problemas com esta abordagem e não é mais considerada uma boa solução. Uma abordagem mais interessante seria representar o tempo implicitamente, isto é, representa-se o tempo pelo efeito que ele tem no processamento e não como uma dimensão adicional da entrada (Elman). Isto significa dar ao sistema de processamento propriedades dinâmicas que são respostas às seqüências temporais. Em resumo, à rede deve ser dada memória.

Esta abordagem pode ser modificada da seguinte forma. Suponha uma rede (mostrada na figura 3.7) aumentada no nível de entrada por unidades adicionais chamadas de Unidades de Contexto. Estas unidades também estão "escondidas" no sentido em que elas interagem exclusivamente com outros nós internos da rede e não com o mundo externo.

Imagine que exista uma entrada seqüencial a ser processada e algum relógio que controle a apresentação da entrada à rede. O processamento então consistiria da seguinte seqüência de eventos. No tempo t, as unidades de entrada recebem a primeira entrada da seqüência. Cada unidade deve ter um valor escalar simples ou um vetor, dependendo da natureza do problema. As unidades de contexto são incialmente colocadas em 0.5[22]. As unidades de entrada e de contexto, ambas, ativam as unidades escondidas; as unidades escondidas, então, alimentam para frente para ativar as unidades de saída. As unidades de saída também retroalimentam para ativar as unidades de contexto. Isto constitui a ativação para frente. Dependendo da tarefa, pode existir ou não uma

[22] A função de ativação usada aqui tem valores entre 0.0 e 1.0.

fase de aprendizado neste ciclo de tempo. Se existir, a saída é comparada à entrada mestre e a propagação para trás do erro é usada para ajustar os pesos de conexão. As conexões recorrentes são fixas em 1.0 e não são sujeitas ao ajuste[23]. No próximo passo de tempo, t+1, a seqüência acima é repetida. Desta vez, as unidades de contexto contêm valores que são exatamente os valores das unidades de saída no tempo t, então estas unidades de contexto provêem a rede de memória.

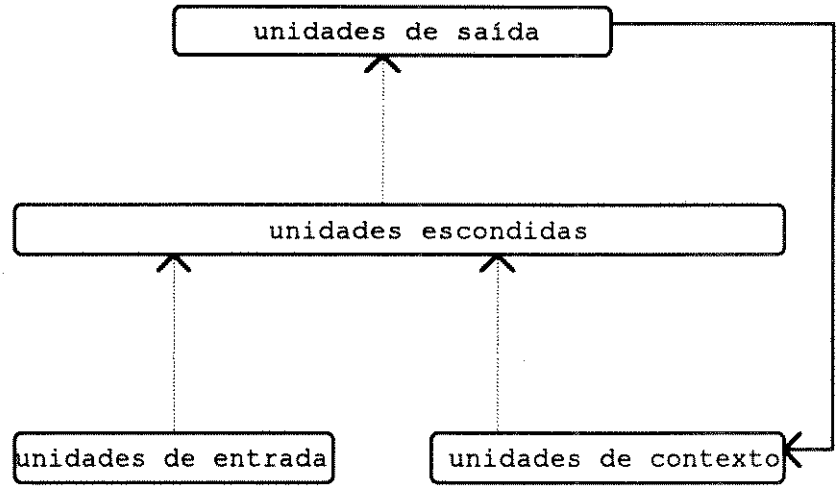


Figura 3.7. Uma rede recorrente simples na qual as ativações são copiadas da camada de saída para a camada de contexto na base um-por-um, com pesos fixos em 1.0. As linhas pontilhadas representam conexões de treinamento

3.3.4.2. Descoberta de classes léxicas a partir da ordem da palavra

Considere um problema que vem do contexto de seqüências de palavras. A ordem das palavras nas sentenças reflete várias restrições. Em linguagens como o Português (também chamadas de linguagens de "ordem fixa de palavras"), a ordem é muito restritiva. A estrutura sintática,

[23] Nas redes usadas aqui, existem conexões um-para-um entre cada unidade de saída e cada unidade de contexto. Isto implica que existe um número igual de unidades de contexto e unidades de saída. As conexões para cima entre as unidades de contexto e as unidades escondidas são totalmente distribuídas, de tal forma que cada unidade de contexto ativa todas as unidades escondidas.

as restrições seletivas, a subcategorização e as considerações de discurso estão entre os muitos fatores que juntos fixam a ordem na qual as palavras ocorrem. Então, a ordem seqüencial das palavras nas sentenças não é nem simples e nem é determinada por uma única causa.

3.3.4.3. Conclusões sobre tarefas temporais

Muitos comportamentos humanos desenvolvem-se através do tempo. Seria tolice tentar entender estes comportamentos sem levar em consideração sua natureza temporal. O conjunto corrente de simulações explora as conseqüências de desenvolver representações do tempo que são distribuídas, dependentes de tarefas e nas quais o tempo é representado implicitamente na dinâmica da rede.

Capítulo 4

Resultados e Conclusões

4.1. Resultados

Vai-se relatar aqui, a seqüência de experimentos realizados e os resultados obtidos, para a implementação do analisador para processamento de texto escrito da língua portuguesa, que consiste de três etapas: análise sintática, análise semântica e análise recorrente. Um maior detalhamento dos resultados pode ser encontrado no Relatório Técnico sobre Resultados, RT - DCA - 07/93. A idéia deste capítulo é mostrar, de forma sucinta, como se implementou este trabalho e as suas saídas.

4.1.1. O analisador sintático

O analisador sintático consiste de uma base de dados lógica, onde estão presentes algumas regras gramaticais da língua portuguesa e um vocabulário restrito. Foi baseado no programa "tokenizador" de C. Marcus (1986) e nas GCDs (Gramáticas de Cláusulas Definidas) de F. Pereira (1980). Esta etapa foi implementada em Prolog.

Foi implementada também a escrita em disco da saída do analisador, chamada de estrutura-chave, que é a frase de entrada só com seus elementos principais (substantivos e verbo), desprezando artigos, adjetivos, etc. O formato é ASCII e o primeiro exemplo ficou assim

[[[homem]],[amar,[[mulher]]]]

para a frase "um homem ama uma mulher". É claro que há a necessidade de "casar" as palavras deste analisador com o semântico e desprezar os caracteres "[", "]" e ",".

Aumentou-se a base de dados do analisador sintático para conter todas as palavras previstas na figura 3.1 (substantivos), no singular e plural, dos determinantes, adjetivos, verbos e preposições da figura 4.1, além da partícula reflexiva *se*, dos nomes próprios *joão* e *maria* e da palavra *com*.

Desta forma, é desejável que os analisadores semântico e recorrente leiam do disco as frases (vai-se manter a opção de se entrar com a frase, quando se desejar). Inseriu-se a construção "com-SN". Manteve-se os adjetivos, mas sem armazená-los na estrutura-chave. Implementou-se nomes próprios, inserindo HOMEM, quando masculino e MULHER, quando feminino. Inseriu-se as preposições "em", "na", "no", etc. (veja figura 4.1) para o verbo *bater*. Considerou-se um valor nulo para preencher os vazios das estruturas (apenas para haver instanciação, senão apareceria a variável não instanciada).

Para o exemplo *Um homem bateu em uma mulher com uma pedra*, ficou:

[[[homem],nulo],[bater,[[[mulher],[[[pedra],nulo]]]]]]

4.1.2. As saídas do analisador sintático

Quando o analisador sintático é executado, a seguinte mensagem aparece no vídeo:

```
Entre com a frase:
```

DETERMINANTES:

todos os	todas as
o/os	a/as
todo	toda
um/uns	uma/umas
nenhum	nenhuma
vários	várias

ADJETIVOS (com seus plurais):

elegante
belo/bela
gostoso/gostosa
forte
frágil
quente
bom/boa
feroz
fresco/fresca

VERBOS:

comer: comeu, come, comerá, comeram, comem, comerão
quebrar: quebrou, quebra, quebrará, quebraram, quebram, quebrarão
mover*: moveu, move, moverá, moveram, movem, moverão
bater: bateu, bate, baterá, bateram, batem, baterão

*Para o verbo *mover*, existem duas formas: a do verbo transitivo direto e a do reflexivo.

PREPOSIÇÕES:

em, no, na, nos, nas

Figura 4.1. Os elementos do analisador sintático.

Ao entrar-se com uma frase entre aspas, o analisador sintático faz a verificação da sua estrutura, baseado nas regras sintáticas que possui e no seu vocabulário e mostra a sua estrutura-chave, ainda no formato inicial, com colchetes e vírgulas. Ao entrar-se com a frase acima (entradas em *itálico*), o resultado será:

```
Entre com a frase: "Um homem bateu em uma mulher com uma pedra".
[[[homem],x],[bater,[[[mulher],[[[pedra],x]]]]]]
Frase CORRETA sintaticamente


Entre com a frase:
```

Um outro exemplo do analisador sintático é a entrada da frase "Todo homem come batatas":

```
Entre com a frase: "Todo homem come batatas".
[[[homem],x],[comer,[[batata],x]]]
Frase CORRETA sintaticamente


Entre com a frase:
```

4.1.3. O analisador semântico

Alguns pontos fundamentais a respeito da implementação:

1. Uma vez analisada sintaticamente uma sentença da língua portuguesa, deve-se acessar uma matriz palavras x microcaracterísticas semânticas, baseada na figura 3.1, onde entra-se com a palavra (um substantivo, um verbo, um objeto) e obtém-se todas as microcaracterísticas (um vetor) para esta palavra.

2. Para cada vetor de microcaracterísticas obtido em (1) alimentar uma "rede" de conexões, com três camadas, onde a entrada seria a conjunção de microcaracterísticas de um único elemento, que chamar-se-á de "estrutura de sentença" e onde a saída seria a conjunção de microcaracterísticas do verbo principal da frase e o substantivo componente (sujeito, objeto ou instrumento), ou a conjunção de microcaracterísticas do objeto com o modificador, que chamar-se-á de "estrutura de caso". Com esta estrutura de caso, o sistema proposto será capaz de reconhecer uma frase, através da ligação entre os seus elementos (sujeito-verbo, verbo-objeto, etc.).

3. Se a palavra não foi "ensinada" ao sistema, pode-se inferir o seu significado, se a mesma constar da tabela de microcaracterísticas (veja figura 3.1).

4. Todos os processos acima devem vir acompanhados por um esquema de fortalecimento ou enfraquecimento dos pesos das conexões (elementos) em cada matriz ("fase do aprendizado").

Fazendo uma retrospectiva do trabalho de McClelland e Kawamoto (1986), chegou-se às seguintes conclusões:

A. O formato da entrada da rede não é a sentença e sim uma representação canônica da sentença. Esta representação canônica são as listas de microcaracterísticas semânticas que representam as palavras da sentença (obtidas no passo 1).

B. Deve haver formas de as palavras da sentença de entrada influenciarem-se. Exemplo:

A menina comeu a boneca.

Aqui, *comeu* é "comeuAVP" (ver tabela 3.2), *menina* e *boneca* são substantivos que preenchem a forma AVP (análise sintática) mas aparentemente a figura 3.1 não tem conexão com a tabela 3.2, portanto não dá para concluir que *boneca* não é alimento e que não deve ser comida. Mas, se observarmos a tabela 3.2, a sentença acima deveria estar entre as sentenças previstas. Não está. Logo, está incorreta semanticamente.

C. Deve-se identificar o verbo da sentença e obter junto a uma matriz (tabela 3.3) todas as microcaracterísticas referentes a este verbo.

D. A partir daí, dá para analisar semanticamente a sentença.

Esta idéia complementa a idéia anterior. Esta implementação, tanto das matrizes, quanto dos algorítmos conexionistas para ajuste de pesos, foi feita em Pascal.

4.1.4. A implementação do algorítmo conexionista

Baseado nas conclusões anteriores, foi implementado um programa em Pascal, que "gera" uma matriz de 20 x 20 baseado no confronto de microcaracterísticas de verbo e substantivos, com pesos que somente são modificados quando se apresenta uma nova sentença à "rede". É lógico que há a necessidade de implementar-se um algorítmo de redes neurais para mexer nas conexões. Também há a necessidade de verificar como proceder na fase de **reconhecimento**, que é quando introduz-se uma frase diferente da ensinada para ver como o modelo se comporta. Deve-se implementar também, o gerador de frases, baseado na tabela 3.2, para treinamento da rede.

Segundo o trabalho de McClelland e Kawamoto, vai-se estudar as estruturas de sentença (ES) e as estruturas de caso (EC). Cada ES tem uma **conexão modificável** para cada EC. Em adição, cada EC tem uma polarização modificável (equivalente a uma conexão a partir de uma unidade que está sempre ligada).

Baseada no padrão ES e nos valores correntes dos pesos, uma entrada da rede para EC é computada, que é a soma dos pesos das entradas ativas para cada unidade mais o termo de polarização. As ECs têm valores de ativação 0 e 1.

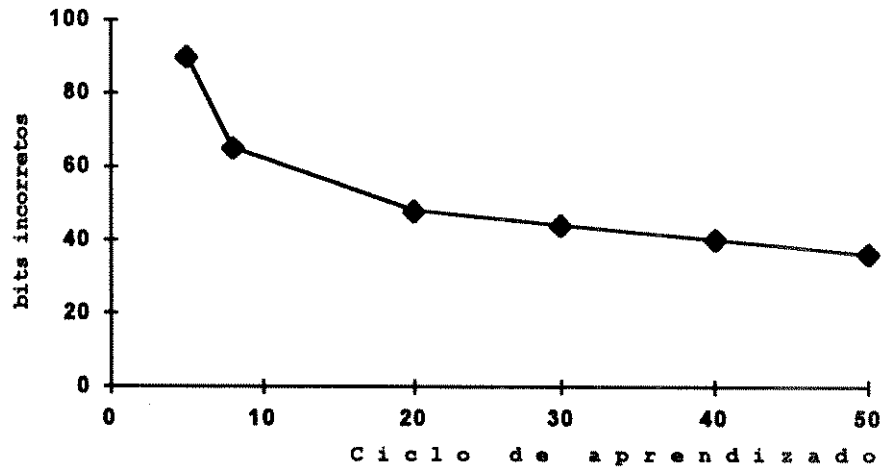


Figura 4.2. Experiência obtida durante o aprendizado (McClelland e Kawamoto, 1986)

4.1.4.1. O número de ciclos necessários

Por outro lado, examinando-se a figura 4.2, obtida do trabalho de McClelland e Kawamoto, conclui-se que (para vetor de 25 microcaracterísticas) para sentenças familiares, após o 20° ciclo de aprendizado, a curva muda pouco (no 20°, a média de bits incorretos é 48, o que significa um erro de 1,92%, pois foram treinadas 2500 unidades de caso, e no 50°, a média é 33, o que significa um erro de 1,32%. Logo, a diferença é muito pequena (0,6%) e, portanto, perfeitamente admissível.) Observe-se que este sistema trata de sentenças da língua inglesa, porém foi observado experimentalmente que isto também é verdade para o sistema implementado. Baseado nisto, vai-se assumir 20 ciclos para a fase de aprendizado da rede.

4.1.4.2. O número de microcaracterísticas semânticas necessárias

Feito mais alguns testes, alterando a iniciação de microcaracterísticas de 25 para 15, por questões de tempo, foram encontradas algumas dificuldades, principalmente nas do verbo. Aí pensou-se que o fato de que a redução de 25 para 15 possa comprometer o desempenho da rede.

Foi testado também para 20 microcaracterísticas. A redução de tempo é considerável, além de reduzir o gargalo: 190 → 25 → 400. Vai-se tentar trabalhar com 20 então, pois a redução é menor devendo comprometer menos o desempenho.

Pensando nas formas de funcionamento da fase de reconhecimento, resolveu-se armazenar em arquivo de disco as duas matrizes de pesos (para AB e para BC) para cada verbo (logo, ter-se-ia 8 arquivos em disco). E, a cada nova frase, ler-se-ia o arquivo correspondente àquele verbo e, atualizar-se-ia a respectiva matriz. Com isso, pode-se utilizar o mesmo conjunto de matrizes de pesos e de reajuste de pesos, para toda frase, óbviamente, reiniciando-as antes.

A fase de reconhecimento dar-se-á quando se desejar descobrir a correção semântica de determinada frase. Entra-se com a frase, busca-se os arquivos correspondentes àquele verbo e a propagação através da rede dá-se apenas uma vez, logo, o processo será extremamente rápido.

4.1.4.3. A velocidade do aprendizado

No procedimento de aprendizado, foi introduzida a variável *alfa*, que segundo E. Rich & K. Knight (1991), aumenta a velocidade de aprendizado. Feito o teste para 20 iterações para três frases no treinamento. Os sistema só acerta se a frase foi treinada. Óbviamente que, para o reconhecimento, deve-se treinar muitas frases. Com o gerador automático de frases, foi feito o teste várias vezes.

4.1.4.4. A relação entre os elementos das frases

Não está havendo relação entre os elementos da frase, ou seja, o sujeito combinado com o verbo está separado do objeto combinado com o verbo, e assim por diante, pois dizem respeito a propagações independentes, matrizes de pesos e correção de pesos diferentes, etc. Será que na frase "O garoto quebrou a janela", o sujeito "garoto" teria o mesmo caso de "O garoto quebrou o vaso"? Será que esta relação é mesmo só entre cada elemento da frase e o verbo? Os elementos não têm ligação entre si?

Veja, por exemplo, a tabela 3.2. Se se considerar apenas ligações dos elementos com o verbo e não com outros elementos, poderia-se aceitar frases como "O humano comeu a presa" e "O animal comeu o alimento com o utensílio" ou "O animal quebrou o objeto_frágil com o quebrador", sentenças estas não permitidas pela tabela 3.2.

O problema torna-se o seguinte: se se gerar as frases como está na tabela 3.2, vai-se repetir estruturas como "homem quebrou", "garoto quebrou", várias vezes. Feitos alguns testes chegou-se à seguinte conclusão: é necessário testar para todas as sentenças previstas na tabela 3.2, mesmo que isso ocasione várias repetições do tipo "homem quebrou" e "garoto quebrou". Se estas repetições ocorrem é porque existem muitas frases possíveis (válidas) com esse início, portanto, a rede deve estar preparada, com maior probabilidade de acertar, para receber estas frases na fase de reconhecimento.

4.1.4.5. Alguns testes realizados

Aumentando-se o número de elementos da camada escondida de 25 para 39, foram realizados alguns testes para cinco sujeitos gerados (*homem, garoto, bola, martelo* e *pedra*). O

resultado obtido foi bem satisfatório. Treinada a rede para os cinco sujeitos mencionados acima em 20 épocas. Depois foi usado como entrada, na fase de reconhecimento, *garoto*, e obtido como diferenças de respostas as seguintes:

0.1 **0.0** 0.1 0.0 **0.1** 0.0 0.0 **0.4** 0.3 0.0
0.0 **0.1** 0.0 0.1 0.0 0.1 0.0 **0.0** 0.0 **0.1**

O único valor um pouco alto é o 0.4. Aumentando o número de ciclos de 20 para 25, obteve-se uma surpresa desagradável:

0.1 0.0 0.1 0.0 0.1 0.0 0.0 0.7 0.6 0.0
0.0 0.1 0.0 0.1 0.0 0.1 0.0 0.0 0.0 0.1

Note que, os valores mais altos da experiência anterior ficaram mais altos ainda: de 0.4 foi para 0.7 e de 0.3 foi para 0.6. A única explicação nisto é o fato de que se houve mais tempo para treinar *garoto*, houve mais tempo para treinar os outros também.

Ampliado a geração para 9 substantivos ao invés de 5 (*homem*, *garoto*, *homem*, *garoto*, *bola*, *martelo*, *pedra*, *vidraca* e *vaso*), o resultado para *garoto*, com 20 ciclos foi:

0.1 0.0 0.1 0.0 **0.2** 0.0 0.0 *0.2* *0.1* 0.0
0.0 0.1 0.0 0.1 0.0 0.1 0.0 **0.1** 0.0 0.1

Como pode-se observar, alguns valores diminuiram (em itálico) e outros aumentaram (em negrito). O fato é que a diferença máxima caiu de 0.4 para 0.2, o que mostra óbviamente, um ótimo resultado.

Para os outros substantivos, não houve uma resposta satisfatória da rede. Talvez haja a necessidade de maior treinamento, uma quantidade maior de vezes, para cada um destes substantivos. Ou talvez, isto é decorrência da diminuição das microcaracterísticas de 25 para 20, que não permite uma melhor distinção entre os substantivos. Outro problema que ainda não foi levado em consideração é a não interligação de elementos na frase, ou seja, o sujeito-verbo não tem relação com o objeto-verbo, e assim por diante.

4.1.4.6. Aumento e redução do número de camadas escondidas

Foi buscada uma alternativa para o desempenho pobre da rede. Foi criada uma outra camada escondida, apesar do desempenho da rede depender do "gargalo" entre a camada escondida e a de saída, mas este "gargalo" manteve-se pois pela limitação do Turbo Pascal, não é possível criar estruturas acima de 64KBytes. Ao testar, percebeu-se que o desempenho piorou, e muito, em alguns casos. Conclusão: não resolveu.

Mas, segundo E. Rich & K. Knight, uma "forma de ajudar a generalização é reduzir o número de unidades escondidas, criando um gargalo entre as camadas de entrada e saída. Confrontada com um gargalo, a rede seria forçada a conseguir representações internas mais compactas de suas entradas". Foram alteradas as saídas da rede de 0.0 para 0.1 e de 1.0 para 0.9, pois como a função de ativação empregada é a sigmóide, deve-se tentar aproximar a saída real com a saída desta função.

Feitos os testes para uma só camada escondida, aconteceram surpresas agradáveis. Testado para 5 substantivos apenas, com 20 ciclos, e fazendo as entradas 0.0 e 1.0.

4.1.4.7. Saídas do analisador semântico

O analisador semântico traz como saída inicial, um "menu" de opções, onde o usuário entra com a opção para executar o analisador semântico (comandos 1 e 2) ou recorrente (comandos 3 e 4). Para o analisador semântico, o comando 1 consiste na fase do aprendizado. Vai-se "ensinar" à rede, todas as palavras que se deseja que a rede conheça. Isto é feito automaticamente pelo gerador de frases.

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
1
Memoria antes = 403536
Memoria depois = 220928

 *** FASE DE TREINAMENTO ***

 Primeira vez? (S/N):
```

Quando se entra com a opção "S" (correspondente a primeira vez que se treina a rede), o analisador semântico começa a treinar a rede para muitas frases geradas pelo gerador. O primeiro verbo a ser treinado é o verbo *bater*. Depois, vem os outros. Para o verbo *bater*, o primeiro sujeito gerado é *homem*. Depois, *menina*, e assim por diante, até que tenham-se esgotadas todas as frases previstas pelo gerador. Então, completa-se uma ativação (uma época). Todas as ativações acontecem para todos os substantivos previstos para o verbo em questão. Note que no início da primeira ativação, todos os valores correspondentes aos pesos de conexões estão próximos de 0.5, pois eles são gerados aleatoriamente entre 0.0 e 1.0. A primeira ativação fornece a seguinte tela:

```
Ativacao 1 - Verbo: Bater - Sujeito: homem

Vetor de saida:

0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.5 0.5 0.5 0.6 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.5 0.6
0.4 0.5 0.6 0.5 0.5 0.6 0.6 0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.5 0.5 0.4 0.5 0.4
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.6 0.5
0.4 0.5 0.5 0.5 0.5 0.6 0.4 0.5 0.5 0.5 0.6 0.4 0.6 0.6 0.5 0.5 0.5 0.5 0.4 0.5
0.6 0.6 0.5 0.6 0.5 0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.4 0.5 0.5 0.5 0.4
0.5 0.6 0.5 0.5 0.6 0.6 0.5 0.4 0.6 0.6 0.5 0.6 0.5 0.6 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.6 0.5 0.5 0.4 0.4 0.6 0.5 0.5 0.4 0.5 0.5 0.5 0.6 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.6 0.5 0.4 0.5 0.5 0.4 0.4 0.5 0.5 0.5 0.5 0.4 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.4 0.5 0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.5 0.5
0.4 0.6 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.5 0.5 0.5 0.6 0.5 0.5 0.5 0.5 0.4 0.5 0.5
0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.6 0.6 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.4 0.6 0.5 0.6 0.5 0.5 0.5 0.5 0.5 0.6 0.4 0.5 0.5 0.5 0.6 0.6 0.5
0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.4 0.4 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.6 0.5 0.6 0.5 0.5 0.5 0.4 0.5 0.4 0.5 0.5 0.6 0.5 0.5 0.5 0.6
0.5 0.6 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.4 0.5 0.4 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.4 0.5 0.5 0.5 0.6 0.5 0.5 0.5 0.6 0.6 0.4 0.4 0.5 0.5 0.5 0.5 0.5
0.5 0.6 0.5 0.6 0.4 0.5 0.5 0.5 0.5 0.4 0.4 0.6 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.4 0.6 0.5 0.4 0.4 0.5 0.5 0.4 0.5 0.6 0.5 0.5 0.6 0.6 0.5 0.4 0.5 0.5
```

Já, para o segundo substantivo gerado, *menina*, já se faz perceber, levemente, que há uma tendência dos valores para o valor final (veja figura 3.1, para *menina*):

```
Ativacao 1 - Verbo: Bater - Sujeito: menina

Vetor de saida:

0.6 0.4 0.6 0.4 0.6 0.4 0.5 0.4 0.6 0.4 0.5 0.6 0.4 0.6 0.4 0.6 0.4 0.4 0.4 0.6
0.4 0.4 0.5 0.4 0.4 0.5 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.3 0.4 0.4 0.4 0.3 0.4 0.4
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.5 0.4 0.4 0.4 0.5 0.5 0.4
0.6 0.4 0.6 0.4 0.6 0.5 0.3 0.4 0.6 0.4 0.4 0.5 0.4 0.6 0.4 0.6 0.4 0.4 0.3 0.6
0.5 0.4 0.4 0.5 0.4 0.4 0.3 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.6 0.5 0.6 0.4 0.6 0.5 0.4 0.4 0.7 0.4 0.4 0.6 0.4 0.6 0.4 0.6 0.4 0.4 0.4 0.6
0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.4 0.4 0.4 0.4 0.5 0.5 0.4 0.4 0.3 0.4 0.4 0.4 0.4 0.3 0.4 0.4 0.4 0.5 0.4 0.4
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.3 0.4 0.4 0.4 0.4 0.4 0.4
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.3 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4
0.5 0.5 0.6 0.4 0.6 0.4 0.4 0.4 0.6 0.4 0.4 0.6 0.4 0.6 0.4 0.6 0.4 0.3 0.4 0.6
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.6 0.4 0.6 0.3 0.6 0.4 0.4 0.4 0.6 0.4 0.4 0.6 0.5 0.5 0.4 0.6 0.4 0.5 0.4 0.6
0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.4 0.4 0.4 0.4 0.5 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.4 0.4 0.5
0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.3 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.6 0.4 0.6 0.4 0.6 0.4 0.4 0.4 0.6 0.4 0.4 0.6 0.5 0.5 0.4 0.6 0.4 0.4 0.4 0.6
0.4 0.5 0.4 0.4 0.3 0.4 0.4 0.4 0.4 0.3 0.4 0.5 0.4 0.4 0.4 0.4 0.4 0.4 0.4 0.4
0.6 0.4 0.5 0.5 0.6 0.4 0.4 0.4 0.6 0.4 0.4 0.6 0.4 0.6 0.4 0.6 0.4 0.3 0.4 0.5
```

Depois de algumas vezes, o programa gera uma saída mais próxima dos valores reais, pois a rede vai corrigindo os pesos das conexões para ficar parecida com a saída desejada:

```
Ativacao 1 - Verbo: Bater - Sujeito: menina

Vetor de saida:

0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1
0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.8 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.1 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.8 0.2 0.8 0.1 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2 0.2
0.8 0.2 0.8 0.2 0.7 0.3 0.2 0.3 0.7 0.2 0.2 0.8 0.2 0.8 0.2 0.8 0.2 0.2 0.2 0.8
```

Quando se digita "N" (ou seja, já "treinou"-se a rede com algumas frases), o analisador semântico pede a próxima estrutura a ser entrada no sistema (este comando permite que se

"treine" a rede aos poucos, pois o treinamento completo pode ser um pouco demorado), e fornece a seguinte tela:

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
1
Memoria antes = 384416
Memoria depois = 201808

 *** FASE DE TREINAMENTO ***

 Primeira vez? (S/N): n

 Qual verbo? (bater/comer/mover/quebrar): comer

 Qual classe? (agente/paciente/instrumento/modificador): agente

Aguarde! Sao 20 ativacoes:
```

Para a opção 2 do "menu" principal (fase do reconhecimento), a rede já "aprendeu" todas as frases previstas para cada verbo e está pronta para reconhecer uma sentença. O sistema pergunta ao usuário se deseja entrar com a frase ou se deseja que o mesmo leia do disco a estrutura-chave da última sentença analisada sintaticamente.

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
2
Memoria antes = 384416
Memoria depois = 201808
 Gostaria de entrar com a frase? (S/N):
```

Caso se digite "N", o sistema vai buscar em disco a estrutura-chave gerada pelo analisador sintático (neste caso, *um homem bateu em uma mulher com uma pedra*):

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
2
Memoria antes = 384416
Memoria depois = 201808
 Gostaria de entrar com a frase? (S/N): n

 *** FASE DE RECONHECIMENTO ***

O sujeito e homem
O verbo e bater
O objeto e mulher
O complemento e pedra
O modificador e pedra
*** LEITURA DO DISCO ***

Verbo: bater - Sujeito: homem

Vetor Media Resultante:
0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.0 0.1 0.9

Vetor Media Desejado:
0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9

Tecle ENTER para continuar...
*** LEITURA DO DISCO ***

Verbo: bater - Objeto: mulher

Vetor Media Resultante:
0.9 0.1 0.9 0.1 0.3 0.7 0.1 0.7 0.3 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9

Vetor Media Desejado:
0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9

Tecle ENTER para continuar...
*** LEITURA DO DISCO ***

Verbo: bater - Complemento: pedra

Vetor Media Resultante:
0.2 0.9 0.4 0.7 0.6 0.4 0.9 0.2 0.2 0.7 0.2 0.4 0.2 0.9 0.2 0.9 0.2 0.4 0.7 0.2

Vetor Media Desejado:
0.1 0.9 0.1 0.9 0.1 0.9 0.9 0.1 0.1 0.1 0.1 0.9 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.1

Tecle ENTER para continuar...
*** LEITURA DO DISCO ***

Verbo: bater - Modificador: pedra

Vetor Media Resultante:
0.1 1.0 1.0 0.1 0.0 1.1 1.1 0.0 0.1 0.1 0.1 1.0 0.1 1.0 0.1 1.0 0.1 1.1 0.1 0.0

Vetor Media Desejado:
0.1 0.9 0.1 0.9 0.1 0.9 0.9 0.1 0.1 0.1 0.1 0.9 0.9 0.1 0.1 0.9 0.1 0.1 0.9 0.1

Tecle ENTER para continuar...
```

Como o sistema não tem condições de distingüir se o complemento é o Instrumento ou o Modificador, então ele trata a palavra como se fosse as duas coisas (veja no exemplo acima a palavra *pedra* é tratada como Complemento (Instrumento) e como Modificador.

A saída do reconhecimento semântico, caso digite-se a opção "S" é a seguinte, com a inclusão da estrutura *homem-quebrar-vidraça-martelo*:

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
2
Memoria antes = 399424
Memoria depois = 216816
 Gostaria de entrar com a frase? (S/N): s

 *** FASE DE RECONHECIMENTO ***


Entre com a frase: Sujeito: homem
Verbo: quebrou
Objeto: vidraca
Complemento: martelo
Modificador:

*** LEITURA DO DISCO ***
```

Neste ponto, o analisador avisa que vai consultar arquivos do disco, onde estão armazenados os pesos das ligações sinápticas da rede para o verbo *quebrar*. Depois de efetuada esta consulta, o analisador fornece a seguinte saída, para o sujeito *homem*:

```
Verbo: quebrou - Sujeito: homem

Vetor Media Resultante:
0.9 0.1 1.0 0.0 0.9 0.1 0.0 0.1 0.9 0.1 0.0 1.0 -0.0 1.0 -0.0 1.0 0.1 0.1 0.0 0.9
Vetor Media Desejado:
0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9

Tecle ENTER para continuar...
```

Para os demais substantivos da frase, o analisador fornece os seguintes resultados:

```
*** LEITURA DO DISCO ***

Verbo: quebrou - Objeto: vidraca

Vetor Media Resultante:
0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9 0.1

Vetor Media Desejado:
0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.9 0.1 0.9 0.1 0.1 0.1 0.9 0.1

Tecle ENTER para continuar...

*** LEITURA DO DISCO ***

Verbo: quebrou - Complemento: martelo

Vetor Media Resultante:
0.1 0.9 0.1 0.9 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.1 0.9 0.9 0.1 0.1 0.1 0.9 0.1

Vetor Media Desejado:
0.1 0.9 0.1 0.9 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.1 0.9 0.1 0.9 0.1 0.1 0.9 0.1

Tecle ENTER para continuar...

*** LEITURA DO DISCO ***

Verbo: quebrou - Modificador:

Vetor Media Resultante:
0.1 0.9 0.1 0.9 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.1 0.9 0.9 0.1 0.1 0.1 0.9 0.1

Vetor Media Desejado:
0.1 0.9 0.1 0.9 0.9 0.1 0.9 0.1 0.1 0.9 0.1 0.1 0.1 0.9 0.1 0.9 0.1 0.1 0.9 0.1

Tecle ENTER para continuar...
```

No caso deste exemplo, como não há modificador, devem ser ignoradas as saídas para o mesmo.

4.1.5. Rede recorrente

Baseado no artigo de Elman sobre Redes Recorrentes (1990), implementou-se uma rede recorrente que faz a "previsão" da ordem de seqüência de palavras e que tem como entrada a seqüência de palavras para cada verbo, segundo as tabelas 3.2 e 3.3, devendo ser capaz de "prever" a próxima palavra na seqüência, e com isso, fazer uma interligação entre os elementos da sentença.

Feitos vários testes para a rede recorrente para "palavras" de 5 bits, portanto, com 5 elementos na camada de entrada, 5 na saída e 20 na escondida e na de contexto. Como não funcionou, resolveu-se tentar outra coisa: ao invés de copiar a camada escondida para a de contexto, como sugere Elman, passou-se a copiar a camada de saída para a de contexto. O resultado foi muito mais satisfatório, com 150 iterações, para 10 "palavras".

Com 200 iterações melhorou ainda mais. O único problema que surge é quando existem duas "palavras" iguais (geradas randomicamente).

Foram feitas modificações no programa de rede recorrente para que funcione com entradas de 20 elementos (as 20 microcaracterísticas dos substantivos). Modificou-se também o gerador de "palavras" para que gerasse as combinações possíveis previstas nas tabelas 3.2 e 3.3. Modificou-se a entrada, para que se entre com a palavra e não com o conjunto de bits, a conversão é, portanto, automática. Fez-se com que a resposta do reconhecimento também fosse automática, ou seja, que o programa faça uma comparação da saída da rede com todas as palavras ensinadas e retorne a palavra que deveria sair e não uma cadeia de bits sem significado.

4.1.5.1. Saídas do analisador recorrente

O treinamento do analisador recorrente, referente ao comando 3, gera toda a seqüência de palavras previstas e treina o modelo durante 30 ativações, com a palavra de entrada, a saída esperada e o vetor de saída real. Como saída inicial, o analisador fornece a seguinte tela:

```
Aguarde! Sao 30 ativacoes:

Verbo - Bater; Entrada: homem
1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0

Saida: homem
1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0

Ativacao 1

Vetor de saida:

0.5 0.5 0.5 0.6 0.5 0.4 0.6 0.5 0.5 0.4 0.5 0.6 0.6 0.6 0.5 0.5 0.7 0.6 0.5 0.5
```

No início da segunda ativação, o analisador começa a repetir toda a seqüência de palavras da primeira ativação. E assim por diante, para as trinta ativações.

Para a fase de reconhecimento (comando 4), o analisador fornece a próxima palavra na seqüência ensinada à rede. Para o caso de *homem*, para o verbo *bater*, o analisador não acha a próxima palavra na seqüência, provavelmente por causa das muitas palavras próximas de *homem* ensinadas ao modelo (tais como *menina*, *garoto*, etc.). A tela inicial é:

```
MENU:
Deseja: 1- aprender palavra
        2- reconhecer palavra
        3- aprender sequencia
        4- reconhecer sequencia
4
Memoria antes = 384416
Memoria depois = 358016

 *** FASE DE RECONHECIMENTO ***

Qual verbo? (bater/comer/mover/quebrar): bater

*** LEITURA DO DISCO ***

Entre com a palavra: homem

homem - Verbo: Bater
1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0

Vetor de saida:

0.0 1.0 0.0 1.0 0.4 0.6 0.5 0.8 0.1 0.9 1.0 0.0 1.0 0.0 1.0 0.0 0.1 0.0 1.0 0.0

SAIDA INVALIDA


Deseja mais alguma entrada?
```

Uma saída válida é a seqüência de *menina*:

```
Deseja mais alguma entrada? s

Entre com a palavra: menina

menina - Verbo: Bater
0.0 1.0 0.0 1.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 0.0 1.0 0.0

Vetor de saida:

0.8 0.2 0.9 0.1 0.4 0.6 0.1 0.6 0.5 0.2 0.2 0.9 0.2 0.8 0.2 0.8 0.3 0.4 0.1 0.9

Saida: homem


Deseja mais alguma entrada?
```

4.1.6. O problema da ambigüidade

Um fato importante que é necessário considerar é a ambigüidade. Quando se entra com a palavra *galinha*, por exemplo, que tem dois significados previstos no analisador semântico, viva e cozida, não há informação explícita sobre seu significado. Como o analisador semântico tratará? (O sintático aceita apenas *galinha*, e tudo bem.) Ele vai alimentar a rede com o conjunto de microcaracterísticas de galinha e a saída dirá se é viva ou cozida.

Outra coisa: como o analisador semântico sabe quando é Instrumento ou Modificador? Pois a frase para ele é passada no formato seqüencial (como no exemplo *um homem bateu em uma mulher com uma pedra*) e a ordem não indica que tipo de complemento é. (No exemplo acima *pedra* é um instrumento, mas como sabê-lo?) Parece que sempre que se tem um "com-SN", tem-se ou um Modificador, aquele que modifica o objeto, ou um instrumento, instrumento da ação do verbo. Mas como identificar? Será que testando nas duas posições possíveis no analisador semântico seria uma saída?

Em relação a este problema, resolveu-se da seguinte forma: como não se sabe se o complemento é Instrumento ou é Modificador, o mesmo é tratado como os dois, ou seja, supõe-se que ele pode ser tanto Instrumento quanto Modificador. Como não é possível existir os dois ao mesmo tempo, a rede dirá se é um ou outro. Acredita-se que essa aproximação está razoável.

4.1.7. Fases do sistema implementado

O sistema implementado para processamento de linguagem natural consiste de três fases:

1ª fase: Análise sintática, feita pelo analisador sintático, construído em Prolog, que fornece como saída a estrutura-chave de uma sentença analisada sintaticamente;

2ª fase: Análise semântica, feita pelo analisador conexionista "backpropagation", que tem como entrada a estrutura-chave da sentença fornecida pelo analisador sintático, ou uma outra frase qualquer, e fornece a informação de correção semântica, baseada no confronto de microcaracterísticas do verbo e dos substantivos que compõem a sentença.

3ª fase: Análise recorrente, feita pelo analisador conexionista recorrente, que receberá a seqüência de palavras e dirá se essa seqüência tem sentido. Para isso, deve-se construir uma rede recorrente para cada verbo e ensinar esta rede com o vetor de microcaracterísticas de cada palavra, para facilitar a generalização. Esta fase também foi implementada com o algorítmo "backpropagation".

Estas três fases compõem a implementação deste trabalho. Vale salientar que esta abordagem mista lógica+conexionismo+recorrência é de certa forma inédita na literatura pesquisada. Espera-se que este trabalho traga alguma contribuição na pesquisa de Processamento de Linguagem Natural para a língua portuguesa, na medida em que mostra caminhos para futuros prosseguimentos de trabalhos dentro deste mesmo tema.

4.2. Conclusões

Este trabalho trouxe como contribuição para o processamento de linguagem natural, uma abordagem mista de lógica e conexionismo. Alcançou-se, com o mesmo, resultados bastante satisfatórios dentro do plano proposto. Espera-se poder continuar o presente trabalho, propondo uma versão com vocabulário e estruturas mais ricos, onde pretende-se que formações mais com-

plexas da língua portuguesa possam ser implementadas. Para isto é necessário aumentar as regras do analisador sintático, juntamente com sua base de dados (fatos). É necessário também, alterar o analisador semântico, construindo redes maiores, onde se permita adequar o tamanho dos vetores de microcaracterísticas semânticas às novas dimensões dadas as palavras, absolutamente necessárias para dar uma diferenciação entre as mesmas. Pode-se, utilizando máquinas mais rápidas, aumentar o número de épocas para treinamento das redes, contribuindo com isso para uma maior eficiência.

4.2.1. A ambigüidade

A **ambigüidade** é a maior causa de incerteza no entendimento de linguagem natural. Um aspecto do problema de ambigüidade é, no caso da ambigüidade **léxica**, resolver qual significado tem uma palavra numa frase dada. Um problema adicional é resolver a ambigüidade **causal** (ou **pragmática**) para inferir a razão mais plausível para alguma ação, de muitas explicações possíveis. Como um exemplo de ambos os problemas, considere a frase (Lange, 1992):

P1: José colocou o pote dentro do lava-louças.

Para entender P1, deve-se entender a palavra *pote* como um "pote de comida", e inferir que a razão de *José* o ter colocado *dentro do lava-louças*, é para lavá-lo. Entretanto, o contexto posterior freqüentemente mostra que as inferências originais podem estar erradas, forçando uma **reinterpretação** da entrada. Este é o caso se P1 for seguido de:

P2: porque a polícia está chegando.

Neste caso, a melhor interpretação para *pote* em P1 muda para "pote de droga", ou alguma outra coisa "proibida", e a ação de *José* parece ser um plano para esconder a droga da polícia e evitar sua prisão. Esta reinterpretação só pode ser feita depois de gerar uma cadeia de inferências para achar o relacionamento causal entre as duas frases. Este é um ponto que pode ser explorado numa continuação deste trabalho. A análise do texto como um todo.

As redes recorrentes também podem ser melhor exploradas num prosseguimento. Pode-se implementar extensões às redes de Elman, previstas por Stolcke (1990), e com isso, ganhar uma melhor performance para sentenças contendo um único predicado principal (codificado por verbos

transitivos ou preposições) aplicadas a objetos de múltiplas características (codificados por sintagmas nominais com modificadores adjetivais) e que mostram robustez contra entradas não-gramaticais.

4.2.2. A implementação

Foi feita uma implementação de um analisador sintático e de um analisador semântico/recorrente. Os resultados obtidos, detalhadamente, assim como as listagens dos programas, encontram-se no Relatório Técnico "Resultados", registrado no Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas, sob número RT-DCA-07/93, de julho de 1993.

4.2.2.1. Limitações do sistema implementado

Na verdade, os analisadores sintático e semântico estão preparados para receber frases formadas por palavras dentro de um contexto geral, porém limitado. Pode-se adequar o sistema para trabalhar num universo mais específico, para isso sendo necessário óbviamente, alterar as palavras que compõem o seu vocabulário. E é intenção também de um sistema futuro, e daí utilizando o conceito de rede recorrente, permitir que se faça o entendimento de textos completos e não apenas frases isoladas.

Bibliografia

1. Albano, E. C. (1989). "Da fala à linguagem tocando de ouvido". IEL - UNICAMP.
2. Araribóia, G. (1988). "Inteligência Artificial - Um curso prático". Livros Técnicos e Científicos Ltda.
3. Backus, J. (1978). "Can Programming Be Liberated from de von Neumann Style? A Functional Style and Its Algebra of Programs". *Communications of the ACM*, August, Volume 21, no. 8, pp. 613-641.
4. Blumberg, A. E. (1976). "Logic - A first course". Alfred A. Knopf.
5. Booker, L. B. & Goldberg, D. E. & Holland, J. H. (1989) "Classifier Systems and Genetic Algorithms". *Artificial Inteligence*, 40, 235-282.
6. Bratko, I. (1986). "Prolog Programming for Artificial Intelligence". Addison-Wesley Publishing Co.
7. Casanova, M. A. & Giorno, F. A. C. & Furtado, A. L. (1987). "Programação em Lógica e a Linguagem Prolog". Editora Edgard Blücher Ltda.
8. Charniak, E. (1981). "The case-slot identity theory". *Cognitive Science*, 5, 285-292.
9. Charniak, E. (1983). "Passing Markers: A Theory of Contextual Influence in Language Comprehension". *Cognitive Science*, 7, 171-190.
10. Chomsky, N. (1972). "Syntactic Structures". Mouton, The Hague - Paris.
11. Collins, A. M. & Loftus, E. F. (1975). "A Spreading-Activation Theory of Semantic Processing" *Psychological Review*, vol. 82, no. 6, 407-428.
12. Colmerauer, A. (1985). "Prolog in 10 figures". *Communications of the ACM*, December, Volume 28, no. 12, pp. 1296-1310.
13. Dahl, V. (1981). "Translating Spanish into Logic through Logic". *American Journal of Computational Linguistics*, Volume 7, Number 3, July-September, pp. 149-164.
14. Dahl, V. (1983). "Logic programming as a representation of knowledge" *IEEE Computer*, October, pp. 106-111.

15. Dascal, M. (1989). "On the Roles of Context and Literal Meaning in Undersatnding". *Cognitive Science* 13, 253-257.
16. Denis, F. A. R. M. & Machado, R. J. (1990). "Introdução aos Sistemas Classificadores e Algorítmos Genéticos". Centro Científico Rio - IBM Brasil.
17. Dubois-Charlier, F. (1976). "Bases de Análise Lingüística" Livraria Almedina.
18. Elman, J. L. (1990). "Finding Structure in Time". *Cognitive Science* 14, 179-211.
19. Fanty, M. A. (1986). "Context-Free Parsing with Connectionist Networks". AIP Conference Proceedings 151, Neural Networks for Computing - Snowbird, UT, Editor: John S. Denker. pp. 140-145.
20. Feldman, J. A. (1982). "Dynamic Connections in Neural Models" *Biological Cybernetics*, 46, 27-39.
21. Feldman, J. A. & Ballard, D. H. (1982). "Connectionist Models and Their Properties". *Cognitive Science* 6, 205-254.
22. Feldman, J. A. (1985). "Connections". *Byte*, pages 277-285, April.
23. Gazdar, G. & Mellish, C. (1989). "Natural Language Processing in Prolog - An Introduction to Computational Linguistics". Addison-Wesley Publishing Company.
24. Geetha, T. V. & Subramanian, R. K. (1990). "Representing Natural Language with Prolog". *IEEE Software*. pages 85-92, vol. 7, no. 2.
25. Grossberg, S. (1987). "Competitive Learning: From Interactive Activation to Adaptive Resonance". *Cognitive Science* 11, 23-63.
26. Hendler, J. A. (1989). "Marker-passing over Microfeatures: Towards a Hybrid Symbolic/Connectionist Model". *Cognitive Science* 13, 79-106.
27. Higginbotham, J. (1984). "Noam Chomsky's Linguistic Theory". In Philosophical Aspects of Artifical Intelligence. Ellis Horwood Limited.
28. Hillis, W. D. (1985). "The Connection Machine". The MIT Press.
29. Hinton, G. E. (1981). "Implementing semantic networks in parallel hardware". In G. E. Hinton & J. A. Anderson (Eds.), Parallel Models of Associative Memory. Hillsdale: Erlbaum.
30. Hinton, G. E. (1992). "How Neural Networks Learn From Experience". *Scientific American*, September, Vol. 267, No. 3, pp. 105-109.
31. Hirst, G. (1991). "Existence Assumptions in Knowledge Representation". *Artificial Intelligence* 49, 199-242.
32. Israel, D. J. (1983). "The role of logic in knowledge". *IEEE Computer*, pages 37-41, October.
33. Iversen, L. L. (1984). "Amino Acids and Peptides: Fast and Slow Chemical Signals in the Nervous System ?". Proceedings of the Royal Society of London, B. volume 221 - pages 245-260 - number 1224 - 22 May 1984.

34. Kandel, E. R. & Schwartz, J. H. (1985). "Principles of Neural Science". Second Edition, Elsevier.
35. Kaufmann, A. (1975). "Introduction to the Theory of Fuzzy Subsets". Volume I, Academic Press.
36. Kohonen, T. (1984). "Self-Organization and Associative Memory". Springer-Verlag.
37. Kolata, G. (1982). "How Can Computers Get Commom Sense". *Science*, Vol. 217, Sept. 24, pages 1237-1238.
38. Kuffler, S. W. & Nicholls, J. G. (1976). "From Neuron to Brain: A Cellular Approach to the Function of the Nervous System". 2nd. Edition. Sunderland.
39. Lange, T. E. (1992). "Lexical and Pragmatic Disambiguation and Reinterpretation in Connectionist Networks". *International Journal of Man-Machine Studies*, 36, 191-220.
40. Lippmann, R. P. (1987). "An Introduction to Computing with Neural Nets". *IEEE ASSP Magazine*, April, pp. 4-22.
41. Lobato, L. M. P. (1986). "Sintaxe Gerativa do Português - Da teoria padrão à teoria da regência e ligação". Editora Vigília Ltda.
42. Luft, C. P. (1989). "Moderna Gramática Brasileira". 9ª edição. Editora Globo.
43. Marcus, C. (1986). "Prolog Programming - Applications for Database Systems, Expert Systems, and Natural Language Systems". Addison-Wesley Publishing Co.
44. Mates, B. (1972). "Elementary Logic". Second edition. Oxford University Press.
45. Mellish, C. S. (1985). "Computer Interpretation of Natural Language Descriptions". Ellis Horwood Limited.
46. McClelland, J. L. & Kawamoto, A. H. (1986). "Mechanisms of Sentence Processing: Assigning Roles to Constituents of Sentences". In Parallel Distributed Processing - Explorations in the Microstructure of Cognition, Volume 2. The MIT Press.
47. McClelland, J. L. & Rumelhart, D. E. (1986). "Parallel Distributed Processing - Explorations in the Microstructure of Cognition". Volume 2 - Psychological and Biological Models. The MIT Press.
48. McClelland, J. L. & Rumelhart, D. E. (1988). "Explorations in Parallel Distributed Processing - A Handbook of Models, Programs, and Exercises". A Bradford Book, The MIT Press.
49. Minsky, M. L. (1980). "K-lines: a theory of memory". *Cognitive Science*, 4, 117-133.
50. Narayanan, A. (1984). "What is it Like to be a Machine ?" In Philosophical Aspects of Artificial Intelligence. Ellis Horwood Limited.
51. Nilsson, N. J. (1982). "Principles of Artificial Intelligence". Springer-Verlag.
52. Nilsson, N. J. (1991). "Logic and Artificial Intelligence" *Artificial Intelligence* 47, 31-56.
53. Obermeier, K. K. (1987). "Natural Language Processing". *Byte*, pages 225-232, December.

54. Pereira, F. C. N. & Warren, D. H. D. (1980). "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks". *Artificial Intelligence* 13, 231-278.

55. Quillian, M. R. (1968). "Semantic Memory". In Semantic Information Processing. Marvin Minsky, Editor. The MIT Press.

56. Rich, E. & Knight, K. (1991). "Artificial Intelligence", Second Edition - International Edition. McGraw-Hill, Inc.

57. Rocha, A. F. (1982). "Basic Properties of Neural Circuits". *Fuzzy Sets and Systems* 7, pp. 109-121.

58. Rocha, A. F. (1990). "Simbolic Reasoning: A Natural Affair for K-Neural Nets". IFLSI Conference, Tizuka, Japan, July.

59. Rosa, J. L. G. & Andrade Netto, M. L. (1993). "Processamento de Linguagem Natural - Uma Abordagem Conexionista". Revista de Informática no. 1, pp. 9-15. Instituto de Informática. Pontificia Universidade Católica de Campinas. Junho.

60. Rumelhart, D. E. & McClelland, J. L. (1986). "Parallel Distributed Processing - Explorations in the Microstructure of Cognition". Volume I - Foundations. A Bradford Book. The MIT Press.

61. Sanfeliu, A. & Alquezar, R. (1992). "Understanding Neural Networks for Grammatical Inference and Recognition". Workshop on Structural and Syntactic Pattern Recognition, Bern, Switzerland, August 26-28.

62. Savadovsky, P. (1988). "Introdução ao Projeto de Interfaces em Linguagem Natural". SID Informática.

63. Schwind, C. B. (1985). "Logic Based Natural Language Processing". In Natural Language Understanding and Logic Programming. V. Dahl and P. Saint-Dizier (Editors), pages 207-219. Elsevier Science Publishers B.V. (North-Holland).

64. Shepherd, G. M. (1974). "The Synaptic Organization of the Brain - An Introduction". Oxford University Press.

65. Soares, A. B. (1990). "Análise Pragmática Baseada em Redes Neuronais Artificiais". Dissertação de Mestrado. Universidade Federal de Uberlândia.

66. Sowa, J. F. (1984). "Conceptual Structures: Information Processing in Mind and Machine". Addison-Wesley Publishing Company.

67. Stolcke, A. (1990). "Learning Feature-based Semantics with Simple Recurrent Networks". International Computer Science Institute - Berkeley, California. TR-90-015, April.

68. Stryer, L. (1975). "Bioquimica". Editorial Reverté, S. A.

69. Swinney, D. A. (1979). "Lexical Access During Sentence Comprehension: (Re)Consideration of Context Effects". *Journal of Verbal Learning and Verbal Behavior* 18, 645-659.
70. Szpakowicz, S. (1987). "Logic Grammars". *Byte*, August, pp. 185-195.
71. Thibadeau, R. & Just, M. A. & Carpenter, P. A. (1982). "A Model of the Time and Content of Reading". *Cognitive Science* 6, 157-203.
72. Touretzky, D. S. & Pomerleau, D. A. (1989). "What's Hidden in the Hidden Layers?" *Byte*, August, pp. 227-233.
73. Vemuri, V. (1988). "Artificial Neural Networks: An Introduction". In Artificial Neural Networks: Theoretical Concepts. V. Vemuri (ed.). The Computer Society of the IEEE. Pp. 1-12.
74. Waltz, D. L. & Pollack, J. B. (1985) "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretations". *Cognitive Science* 9, 51-74.
75. Warner, A. J. (1988). "Natural Language Processing in Information Retrieval". *Bulletin of the American Society for Information Science*, pages 18-19, August/September.
76. Webber, B. L. (1983). "Logic and Natural Language". *IEEE Computer*, pages 43-46, October.
77. Wilks, Y. (1975). "A preferential, pattern-seeking semantics for natural language inference". *Artificial Intelligence*, 6, 53-74.
78. Woods, W. A. (1970). "Transition Network Grammars for Natural Language Analysis". *Communications of the ACM*, volume 13, no. 10, October, pp. 591-606.
79. Woods, W. A. (1983). "What's important about knowledge representation?". *IEEE Computer*, pages 22-27, October.
80. Zadeh, L. A. (1965). "Fuzzy Sets". *Information and Control* 8, 338-353.
81. Zeidenberg, M. (1987). "Modelling the Brain". *Byte*, pages 237-246, December.