SISTEMAS BASEADOS EM CONHECIMENTO

João Luís Garcia Rosa
Instituto de Informática
PUC-Campinas

joaol@ii.puc-campinas.br
www.ii.puc-campinas.br/joaoluis

I. INTRODUÇÃO

1.1. Histórico - Sistemas Baseados em Conhecimento (Russell e Norvig, 1995)

- Buchanan (1969): DENDRAL: usa conhecimento para inferir estrutura molecular a partir da informação de um espectômetro de massa: grande número de regras de propósito geral
- Feigenbaum: Heuristic Programming Project
- Sistemas especialistas: MYCIN com 450 regras
- Duda (1979): Prospector: prospeção de molibdênio
- Entendimento de Línguas Naturais:
  - SHRDLU de Winograd: ainda dependência da sintaxe
  - Charniak: conhecimento geral sobre o mundo
  - Woods (1973): LUNAR: Processamento de Linguagem Natural (PLN) como interfaces para Banco de Dados.
- McDermott (1962): 1o. Sistema especialista comercial bem sucedido: R1 na DEC: Em 1986, economia de US$40 milhões
- 1981: Japão anuncia o projeto da "Quinta Geração": Prolog, inferência lógica e PLN
- Indústrias:
  - Software: Carnegie Group, Inference, Intellicorp, Teknowledge
  - Hardware: Lisp Machines Inc., Texas Instruments, Symbolics, Xerox
  - Vendas: de poucos milhões em 1980 para 2 bilhões em 1988.

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

Em ciência da computação, uma boa solução depende de uma boa representação. Para a maioria das aplicações em Inteligência Artificial (IA), a escolha da representação é mais difícil, pois as possibilidades são maiores e os critérios são menos claros.

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

- como estruturar um sistema de representação que será capaz, em 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;
- como assimilar partes do conhecimento, na ordem em que elas são encontradas, ao invés de uma ordem específica de apresentação.

A IA deve ter mecanismos para a representação de fatos. A abordagem mais comum para este fim é usar a linguagem da *lógica*. A prova de teoremas foi um dos primeiros domínios nos quais as técnicas de IA eram exploradas.

Usa-se para a representação de conhecimento através da lógica os seguintes símbolos lógicos padrões: "$\rightarrow$" (implicação), "$\neg$" (negação), "$\vee$" (disjunção), "$\wedge$" (conjunção), "$\forall$" (quantificação universal = "para todos") e "$\exists$" (quantificação existencial = "existe").

1.3. As abordagens simbólica e sub-simbólica da IA

A Inteligência Artificial trabalha basicamente com duas abordagens: a abordagem *simbólica*, baseada na lógica, e a abordagem *sub-simbólica* ou *conexionista*, baseada na propagação de processadores elementares (as redes neurais artificiais), ou seja, simulação do cérebro humano. Faz parte da abordagem simbólica os sistemas baseados em regras e os sistemas especialistas. Na mistura das duas abordagens surgiram as chamadas redes neurais baseadas em conhecimento.

II. LÓGICA DE PREDICADOS PARA REPRESENTAÇÃO DO CONHECIMENTO

2.1. Lógica Sentencial ou Cálculo Proposicional

A motivação para se estudar lógica num curso de Sistemas Baseados em Conhecimento é de usar esta linguagem artificial como uma ferramenta para fazer dedução lógica em base de conhecimento.

Para representar o conhecimento de mundo que um sistema de IA necessita, usa-se a lógica proposicional. Vai-se representar os fatos do mundo real através das proposições lógicas:

Está chovendo: *CHOVENDO*
Está ensolarado: *ENSOLARADO*
Se está chovendo, então não está ensolarado:
CHOVENDO $\rightarrow \neg$*ENSOLARADO*

2.1.1. Sintaxe das linguagens proposicionais

Dado um alfabeto α, uma *cadeia* sobre α é uma seqüência de símbolos de α. Uma *linguagem* sobre α é um conjunto de cadeias de α.

$\otimes$[2] Um *alfabeto proposicional* α consiste de

símbolos lógicos:
pontuação: (,)
conectivos: $\neg$ (negação)
$\wedge$ (conjunção)
$\vee$ (disjunção)
$\rightarrow$ (implicação)
$\leftrightarrow$ (bi-implicação ou bi-condicional)

[2] O símbolo $\otimes$ será usado para representar uma definição.

símbolos não-lógicos: um conjunto finito **P** de *símbolos proposicionais* diferentes dos símbolos lógicos. Ex. P, Q, etc.

Todas as definições a seguir têm por base o alfabeto proposicional α.

⊗ O conjunto de *fórmulas proposicionais* (ou simplesmente *fórmulas*) é o menor conjunto de cadeias satisfazendo às seguintes condições:

i. todo símbolo proposicional é uma fórmula;
ii. se P e Q são fórmulas, então $(\neg P)$, $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$ e $(P \equiv Q)$ também são fórmulas.

⊗ Uma fórmula Q é uma *subfórmula* de uma fórmula P se e somente se Q é uma subcadeia de P. Em outras palavras, toda subcadeia de P que ainda for uma fórmula é uma subfórmula.

⊗ A *linguagem proposicional*, denotada por $L(\alpha)$, é o conjunto das fórmulas proposicionais.

2.1.2. Semântica das linguagens proposicionais

As fórmulas de uma linguagem proposicional, incluindo os símbolos proposicionais, terão como significado os valores-verdade *FALSO* ou *VERDADEIRO*, abreviados *F* e *V*, respectivamente.

⊗ Seja **P** o conjunto de símbolos proposicionais de α. Uma *atribuição de valores-verdade* para α (ou simplesmente uma *atribuição* para α) é uma função $a: \mathbf{P} \rightarrow \{F,V\}$.

⊗ Seja *a* uma atribuição de valores-verdade. A *função de avaliação* para $L(\alpha)$ *induzida* por *a* é a função v: $L(\alpha) \rightarrow \{F,V\}$ definida da seguinte forma (Obs: note que a *atribuição* é para símbolos proposicionais e a *avaliação* é para fórmulas):

$v(A) = a(A)$, se A é um símbolo proposicional

$v(\neg P) = V$, se $v(P) = F$
$= F$, se $v(P) = V$

$v(P \wedge Q) = V$, se $v(P) = v(Q) = V$
$= F$, em caso contrário

$v(P \vee Q) = F$, se $v(P) = v(Q) = F$
$= V$, em caso contrário

$v(P \rightarrow Q) = F$, se $v(P) = V$ e $v(Q) = F$
$= V$, em caso contrário

$v(P \leftrightarrow Q) = V$, se $v(P) = v(Q)$
$= F$, em caso contrário

⊗ Sejam **P** e **Q** conjuntos de fórmulas em $L(\alpha)$ e R uma fórmula em $L(\alpha)$.

(a) R é *verdadeira* em uma atribuição de valores-verdade *a* se e somente se $v(R) = V$. Em caso contrário, R é *falsa*.
(b) R é uma *tautologia* se e somente se, para toda atribuição de valores-verdade *a*, $v(R) = V$. Isso corresponde a uma coluna inteira de *V* (verdadeiro) na tabela-verdade.
(c) uma atribuição de valores-verdade *a satisfaz* a **P**, ou *a* é um *modelo* para **P**, se e somente se, para toda fórmula S em **P**, $v(S) = V$. Isso corresponde a uma linha inteira de *V* (verdadeiro) na tabela-verdade.
(d) **P** é *satisfatível* se e somente se existe uma atribuição de valores-verdade *a* que satisfaz **P**. Em caso contrário, **P** é *insatisfatível*.

(e) R é uma *conseqüência lógica* de **P**, ou **P** *implica logicamente* R (notação: **P** |= R), se e somente se, para toda atribuição de valores-verdade *a*, se *a* satisfaz **P** então *a* satisfaz R.

(f) **P** é *tautologicamente equivalente* a **Q** (notação: **P** |=| **Q**) se e somente se toda fórmula de **Q** for conseqüência lógica de **P** e vice-versa.

2.1.3. O Método da tabela-verdade

O método da tabela-verdade baseia-se na observação de que, se **P** é um conjunto finito de fórmulas e Q é uma fórmula, há um número finito de símbolos proposicionais ocorrendo em Q e nas fórmulas em **P**. Logo, há um número finito de atribuições de valores-verdade distintas para estes símbolos. Assim, para decidir se Q é uma conseqüência lógica de **P**, basta enumerar todas estas atribuições e, para cada uma delas que satisfizer a todas as fórmulas em **P**, testar se ela também satisfaz a Q. Se esta condição for sempre observada, então pode-se afirmar que Q é uma conseqüência lógica de **P**. Note que, se *a* não satisfizer a alguma fórmula em **P**, o valor que *a* atribui a Q não importará, pela forma como a implicação lógica foi definida. Este método também pode ser usado para decidir se uma fórmula é satisfatível ou se dois conjuntos finitos de fórmulas são tautologicamente equivalentes.

Exemplo: Seja **P** o seguinte conjunto de fórmulas:

1. $A \rightarrow \neg B$
2. $B \wedge A$
3. B,

seja **Q** o seguinte conjunto de fórmulas:

1. $A \vee B$
2. $B \rightarrow A$

e seja R a fórmula $\neg B \rightarrow A$

vamos construir a tabela-verdade para estes conjuntos

		1	2	3	4	5	6
A	B	$A \rightarrow \neg B$	$B \wedge A$	B	$A \vee B$	$B \rightarrow A$	$\neg B \rightarrow A$
F	*F*	*V*	*F*	*F*	*F*	*V*	*F*
F	*V*	*V*	*F*	*V*	*V*	*F*	*V*
V	*F*	*V*	*F*	*F*	*V*	*V*	*V*
V	*V*	*F*	*V*	*V*	*V*	*V*	*V*

Observe, que para o conjunto **P** (colunas 1, 2 e 3), não existe nenhuma atribuição (linha da tabela-verdade) que o satisfaça, ou seja, não existe nenhuma atribuição que resulta sempre em verdadeiro para **P**. Logo, **P** é insatisfatível. Já para o conjunto **Q** (colunas 4 e 5), existem duas atribuições (A=*V*, B=*F* e A=*V*, B=*V*) que satisfazem este conjunto, logo **Q** é satisfatível. Como a fórmula R (coluna 6) também é verdadeira para as duas atribuições que satisfazem **Q**, diz-se que R é conseqüência lógica de **Q**, ou que **Q** implica logicamente R. Observe também que **P** e **Q** não são tautologicamente equivalentes.

2.2. Lógica de Primeira Ordem

A Lógica de primeira ordem, ou Cálculo de Predicados de Primeira Ordem (CPPO) pode ser caracterizada como um sistema formal apropriado a definição de teorias do universo de discurso da Matemática. A motivação para se estudar esta lógica é que a lógica sentencial não dá conta da representação de frases do tipo:

"Todos os homens são mortais."
$\forall x$ (HOMEM(x) $\rightarrow$ MORTAL(x))

Tome uma base de conhecimento com sentenças em linguagem natural. O primeiro passo é identificar os possíveis predicados, que são relações entre objetos. Verbos sempre são predicados. Substantivos também são, desde que se refira a classe de objetos. Indivíduos são representados por constantes. Apenas predicados têm

variáveis que devem ser quantificadas. Depois deve-se tentar ler este conhecimento como se fosse um condicional, para então representá-lo através de uma implicação lógica, relacionando os seus predicados usando os conectivos lógicos:

Todos os homens são mortais
Se x é homem então x é mortal
$\forall x\ (HOMEM(x) \rightarrow MORTAL(x))$
Sócrates é homem
HOMEM(sócrates)
Todo homem ama uma mulher
Se x é homem e y é mulher então x ama y
$\forall x\, \exists y\ ((H(x) \wedge M(y)) \rightarrow A(x,y))$ ou
$\exists y\, \forall x\ ((H(x) \wedge M(y)) \rightarrow A(x,y))$

2.2.1. Sintaxe das linguagens de primeira ordem

⊗ Um *alfabeto de primeira ordem* α consiste de:

símbolos lógicos:
pontuação: (,)
conectivos: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$
quantificadores: $\forall$ (quantificador universal)
$\exists$ (quantificador existencial)
variáveis: um conjunto de símbolos distintos dos demais, por convenção, representadas por letras minúsculas do final do alfabeto: *x*, *y*, *z*, etc.
símbolo de igualdade (opcional): =

símbolos não-lógicos: Um conjunto, possivelmente vazio, de *constantes* distintas dos demais símbolos, por convenção, representadas por letras minúsculas do início do alfabeto: a, b, c, etc.

Para cada n > 0, um conjunto, possivelmente vazio, de *símbolos funcionais n-ários* distintos dos demais símbolos (O símbolo funcional representa a função da computação, que efetua um cálculo e retorna um valor), por convenção, representados pelas letras minúsculas a partir da letra f (de funcional): f, g, h, etc.

Para cada n > 0, um conjunto, possivelmente vazio, de *símbolos predicativos n-ários* distintos dos demais símbolos (O símbolo predicativo, ou *predicado*, representa uma relação entre objetos, ou seja sua avaliação será verdadeira ou falsa), por convenção, representados por letras maiúsculas: P, Q, R, etc.

Todas as definições a seguir fazem referência ao alfabeto de primeira ordem α.

⊗ O conjunto de *termos de primeira ordem* (ou simplesmente *termos*) é o menor conjunto satisfazendo às seguintes condições:

i. toda variável é um termo;
ii. toda constante é um termo;
iii. se $t_1,...,t_n$ são termos e f é um símbolo funcional n-ário, então $f(t_1,...,t_n)$ também é um termo.

⊗ O conjunto de *fórmulas* é o menor conjunto satisfazendo às seguintes condições:

i. se $t_1,...,t_n$ são termos e P é um símbolo predicativo n-ário, então $P(t_1,...,t_n)$ é uma fórmula, chamada de *fórmula atômica*. (Observe que os argumentos do predicado são termos, ou seja, podem ser constantes, variáveis ou símbolos funcionais, mas *nunca* outros predicados).
ii. se $t_1,...,t_n$ são termos e "=" é um símbolo de α, então $(t_1 = t_2)$ é uma fórmula, também chamada de *fórmula atômica*.
iii. se P e Q são fórmulas, então $(\neg P)$, $(P \wedge Q)$, $(P \vee Q)$, $(P \rightarrow Q)$ e $(P \leftrightarrow Q)$ também são fórmulas.

iv. se P é uma fórmula e x é uma variável, então $\forall x$(P) e $\exists x$(P) também são fórmulas.

⊗ Uma fórmula Q é uma *sub-fórmula* de uma fórmula P se e somente se Q é uma subcadeia de P.

⊗ A *linguagem de primeira ordem*, denotada por L(α), é o conjunto de termos e fórmulas de primeira ordem.

⊗

(a) Em uma fórmula da forma $\forall x$(Q) (ou da forma $\exists x$(Q)), Q é o *escopo* de $\forall x$ (ou de $\exists x$).
(b) Uma ocorrência de uma variável x em uma fórmula P é *ligada* em P, se a ocorrência se dá em uma subfórmula de P da forma $\forall x$(Q) ou da forma $\exists x$(Q). Caso contrário, a ocorrência de x é *livre*. Ou seja, a ocorrência de x é ligada se x ocorrer dentro do escopo de seu quantificador.
(c) Uma variável x é *livre* em P se existe uma ocorrência livre de x em P.
(d) Uma fórmula P é uma *sentença* se e somente se nenhuma variável ocorre livre em P. Por exemplo, $\forall x$ (A(x) $\wedge$ B(y)) $\rightarrow$ C(x) não é uma sentença, pois a variável y ocorre livre (não existe quantificador para ela, e a segunda ocorrência de x também é livre (está fora do escopo de $\forall x$).

⊗

(a) Dada uma fórmula P, com variáveis livres x_1,...,x_n, o *fecho universal* de P é a fórmula $\forall x_1 ... \forall x_n$ (P) e o *fecho existencial* de P é a fórmula $\exists x_1 ... \exists x_n$ (P).
(b) Uma fórmula P está na *forma normal prenex* se e somente se P for da forma Q(M) onde Q, o *prefixo* de P, é uma cadeia de quantificadores e M, a *matriz* de P, é uma fórmula sem ocorrências de quantificadores. Ou seja, forma normal prenex é quando se tem todos os quantificadores à esquerda da fórmula. Por exemplo, a fórmula, que é uma sentença, $\forall x \exists y$ (A(x,y) $\rightarrow$ B(x)) está na forma normal prenex, mas a fórmula, que também é uma sentença, $\forall x$ A(x) $\rightarrow \exists y$ B(y) não está.
(c) Uma fórmula P é uma *conjunção* se e somente se, omitindo-se os parênteses, for da forma $P_1 \wedge ... \wedge P_n$.
(d) Uma fórmula P é uma *disjunção* se e somente se, omitindo-se os parênteses, for da forma $P_1 \vee ... \vee P_n$.
(e) Uma fórmula P está em *forma normal conjuntiva* se e somente se estiver em forma normal prenex e a sua matriz for uma conjunção de disjunções de fórmulas atômicas, negadas ou não. Por exemplo, $\forall x \exists y$ ((A(x,y) $\vee$ B(x)) $\wedge$ (B(y) $\vee$ A(y,y)), está na forma normal conjuntiva. E $\forall x \exists y$ (A(x,y) $\vee$ B(x)) também está. E $\forall x \exists y$ (A(x,y) $\wedge$ B(y)) também, assim como $\forall x \exists y$ (A(x,y)). Por que ?

2.3. Notação Clausal

⊗

(a) Um *literal positivo* é uma fórmula atômica.
(b) Um *literal negativo* é a negação de uma fórmula atômica.
(c) Um *literal* é ou um literal positivo ou um literal negativo.
(d) Dois literais têm *sinais opostos* se e somente se um deles for positivo e o outro for negativo.
(e) Dois literais são *complementares* se e somente se um deles for a negação do outro.
(f) Uma fórmula atômica F é o *átomo* de um literal L, denotado por |L|, se e somente se L for F ou $\neg F$.

⊗ Uma *cláusula* é ou uma seqüência não vazia de literais ou a *cláusula vazia*, denotada por " ".

⊗ A *linguagem de cláusulas* é o conjunto de todas as cláusulas.

⊗ Uma interpretação I *satisfaz* uma cláusula não vazia C (denotado por I |= C) se e somente se I satisfaz a sentença F definida como

$$\forall x_1 ... \forall x_m (L_1 \vee ... \vee L_n)$$

onde x_1 ,...,x_m são as variáveis ocorrendo em C e L_1 ,...,L_n são os literais de C. Diz-se ainda que C e F são *equivalentes*. Por convenção, a cláusula vazia é sempre insatisfatível. Com esta definição, está-se querendo dizer que, apesar da cláusula, por definição, ser uma seqüência de literais, estes literais estão ligados através de disjunções, muito embora alguns autores não as tornem explícitas na sua representação. Pode-se dizer então, que uma *cláusula é uma disjunção de literais*.

2.3.1. Representação clausal de fórmulas

⊗ Um conjunto de cláusulas **S** é uma *representação clausal* para uma fórmula P se e somente se P é satisfatível se e somente se **S** é satisfatível.

A obtenção da representação clausal de uma fórmula é um processo mecânico, como descrito a seguir:

Algoritmo de Representação Clausal

entrada: uma fórmula P
saída: uma representação clausal **S** para P

(a) Tome o fecho existencial de P

Se P contiver uma variável livre *x*, substitua P por $\exists x$(P). Repita este passo até que a fórmula não tenha variáveis livres.

(b) Elimine quantificadores redundantes

Elimine todo quantificador "$\forall x$" ou "$\exists x$" que não contenha nenhuma ocorrência livre de *x* no seu escopo. (Ou seja, elimine todo quantificador desnecessário).

(c) Renomeie variáveis quantificadas mais de uma vez.

Se houver dois quantificadores governando a mesma variável, substitua a variável de um deles e todas as suas ocorrências livres no escopo do quantificador por uma nova variável que não ocorra na fórmula. Repita o processo até que todos os quantificadores governem variáveis diferentes.

(d) Elimine os conectivos "$\rightarrow$" e "$\leftrightarrow$"

Substitua:

$(Q \rightarrow R)$	por	$(\neg Q \vee R)$
$(Q \leftrightarrow R)$	por	$(\neg Q \vee R) \wedge (Q \vee \neg R)$
$\neg(Q \rightarrow R)$	por	$(Q \wedge \neg R)$
$\neg(Q \leftrightarrow R)$	por	$(Q \wedge \neg R) \vee (\neg Q \wedge R)$

(e) Mova "¬" para o interior da fórmula

Até que cada ocorrência de "¬" preceda imediatamente uma fórmula atômica, substitua:

$\neg\forall x(Q)$	por	$\exists x(\neg Q)$
$\neg\exists x(Q)$	por	$\forall x(\neg Q)$

$\neg(Q \wedge R)$	por	$(\neg Q \vee \neg R)$
$\neg(Q \vee R)$	por	$(\neg Q \wedge \neg R)$

$\neg\neg Q$	por	Q

(f) Mova os quantificadores para o interior da fórmula (opcional)

Substitua, caso *x* não seja livre em R,

$\forall x\,(Q \vee R)$	por	$\forall x\,(Q) \vee R$

$\forall x\,(R \vee Q)$	por	$R \vee \forall x(Q)$
$\forall x\,(Q \wedge R)$	por	$\forall x\,(Q) \wedge R$
$\forall x\,(R \wedge Q)$	por	$R \wedge \forall x(Q)$
$\exists x\,(Q \vee R)$	por	$\exists x(Q) \vee R$
$\exists x\,(R \vee Q)$	por	$R \vee \exists x(Q)$
$\exists x\,(Q \wedge R)$	por	$\exists x(Q) \wedge R$
$\exists x\,(R \wedge Q)$	por	$R \wedge \exists x(Q)$

(g) Elimine os quantificadores existenciais

Seja Q a fórmula corrente. Crie a nova fórmula corrente substituindo a subfórmula de Q da forma "$\exists y$(R)", que se situa mais à esquerda, por "R[y/f(x_1,...,x_n)]", onde x_1,...,x_n é uma lista de todas as variáveis livres de "$\exists y$(R)" e "f" é qualquer símbolo funcional n-ário que não ocorre em Q. Quando não houver variáveis livres em "$\exists y$(R)", substitua "$\exists y$(R)" por "R[y/c]", onde "c" é uma constante que não ocorre em Q. Repita o processo até que todos os quantificadores existenciais tenham sido eliminados.

Na verdade, deve-se pegar o escopo do quantificador existencial e verificar se, além da variável quantificada existencialmente, existe alguma variável ocorrendo livre neste escopo? Se existir, substitui-se a variável existencial por uma função de todas as variáveis que ocorrem livre. Se não, substitui-se a variável existencial por uma constante. A explicação para este procedimento pode ser entendida através do seguinte exemplo. Para a sentença ambígua da linguagem natural

"Todo homem ama uma mulher".

Esta sentença tem pelo menos duas leituras possíveis. Numa, cada homem ama a sua mulher, portanto para cada homem (são todos) existe uma mulher que ele ama. Já numa outra leitura, existe uma única mulher, que é representada por uma constante a, que todos os homens amam. Logo as leituras gerariam as seguintes fórmulas:

$$\forall x \exists y\,((H(x) \wedge M(y)) \rightarrow A(x, y))$$
$$\exists y \forall x\,((H(x) \wedge M(y)) \rightarrow A(x, y))$$

Observe que a diferença das interpretações está apenas nos escopos dos quantificadores. A primeira fórmula gerará uma representação, onde y será substituída por um f(x), ou seja, cada homem tem a sua mulher, portanto y que é mulher é função de x, que é homem. Na segunda fórmula o y será substituído por uma constante, pois é a mesma mulher que todos os homens amam.

Exemplo. Na fórmula $\exists x \forall y\,((P(x) \wedge Q(y))$, substitui-se a variável x por uma constante a, pois no escopo do quantificador existencial, que é $\forall y\,((P(x) \wedge Q(y))$, nenhuma variável além do x ocorre livre. Logo, a fórmula fica $\forall y\,((P(a) \wedge Q(y))$. Já na fórmula $\forall y \exists x\,(P(x) \vee Q(y))$, substitui-se x por $f(y)$, pois no escopo do quantificador existencial que é $(P(x) \vee Q(y))$, além do x, o y ocorre livre, logo substitui-se a variável x por uma função da variável livre y, ou seja, $f(y)$. Assim, a fórmula fica $\forall y\,(P(f(y)) \vee Q(y))$.

Os novos símbolos funcionais assim introduzidos são chamados de *funções de Skolem* e o processo de substituição é chamado de *Skolemização*.

(h) Obtenha a forma normal prenex

Mova os quantificadores universais para a esquerda.

(i) Obtenha a forma normal conjuntiva

Até que a matriz da fórmula seja uma conjunção de disjunções, substitua:

$(Q \wedge R) \vee S$	por	$(Q \vee S) \wedge (R \vee S)$
$Q \vee (R \wedge S)$	por	$(Q \vee R) \wedge (Q \vee S)$

(j) Simplifique (opcional)

Transforme a fórmula Q resultante do passo (i) em outra mais simples S tal que S ainda esteja em forma normal conjuntiva (sem quantificadores existenciais) e Q seja satisfatível se e somente se S o for.

(k) Obtenha a representação clausal

A representação clausal **S** da fórmula inicial P será o conjunto das cláusulas da forma $L_1 ... L_n$ tais que $L_1 \vee ... \vee L_n$ é uma disjunção da matriz da fórmula resultante do passo anterior. Cada cláusula deve ficar numa linha (disjunção de literais) e cláusulas em linhas diferentes pressupõe conjunção de cláusulas.

III. REPRESENTAÇÃO DO CONHECIMENTO E INFERÊNCIA LÓGICA

3.1. Resolução

3.1.1. Unificação

⊗

(a) Um par (x,t) é uma *substituição simples* (lê-se "*x* substituído por *t*") se e somente se *x* é uma variável e *t* é um termo.
(b) Um conjunto finito β de substituições simples é uma *substituição* se e somente se duas substituições simples em β não coincidem no primeiro elemento.
(c) Uma substituição β é uma *substituição básica* se e somente se, para todo (x,t) em β, *t* é um termo sem ocorrências de variáveis.
(d) β é a *substituição vazia* se e somente se β for o conjunto vazio.
(e) Uma substituição β é uma *renomeação de variáveis* ou, simplesmente, uma *renomeação*, se e somente se cada par (x,t) em β for tal que *t* é uma variável e não existirem dois pares (x,u) e (y,v) em β tais que $x \neq y$ e $u = v$.

A expressão "*x*/*t*" denotará uma substituição simples (x,t). Letras gregas denotarão substituições e, em especial, "ε" denotará a substituição vazia.

Uma *expressão* é qualquer seqüência de símbolos de um alfabeto de primeira ordem, e uma *expressão simples* é qualquer literal ou termo sobre o alfabeto.

⊗ Seja C uma cláusula e β uma substituição. A *instanciação* de C por β, denotada por Cβ, é a cláusula obtida instanciando-se C por β e eliminando-se as ocorrências repetidas do mesmo literal, exceto a ocorrência mais à esquerda.

⊗ A *composição* de substituições é a função, denotada por "∘", que mapeia pares de substituições em uma substituição e é definida da seguinte forma:

Para todo par de substituições

$\beta = \{x_1 / t_1 ,..., x_n / t_n , y_1 / s_1 ,..., y_k / s_k \}$
$\theta = \{y_1 / r_1 ,...,y_k / r_k , z_1 / q_1,..., z_m / q_m \}$

onde $x_1,..., x_n , y_1,..., y_k , z_1,..., z_m$ são variáveis distintas, a composição de β com θ será a substituição:

$\beta \circ \theta = \{x_1 /(t_1)\theta,..., x_n /(t_n)\theta, y_1 /(s_1)\theta,..., y_k /(s_k)\theta, z_1 / q_1,..., z_m / q_m \}$

⊗ Seja C = L_1 L_2 ... uma cláusula com conjunto de literais **L** = {L_1, L_2} e β uma substituição.

(a) β é um *unificador* para **L** se e somente se $(L_1)\beta = (L_2)\beta$.
(b) β é um *unificador mais geral* (ou, abreviadamente, um *u.m.g.*) de **L** se e somente se β é um unificador de **L** e, para todo unificador θ de **L**, existe uma substituição φ tal que $\theta = \beta \circ \varphi$.
(c) conjunto **L** é *unificável* se e somente se existe um unificador para **L**.

3.1.2. O Algoritmo da Unificação

⊗ Um conjunto de termos **D** é o *conjunto de discórdia* de um conjunto de literais **L** = {L_1 ,...,L_n } se e somente se

i. **D** = ∅, se $n = 1$;
ii. **D** = {t_1 ,...,t_n }, se $n > 1$ e todas as expressões em **L** são idênticas até o *i*-ésimo símbolo, exclusive, e t_j é o termo ocorrendo em **L** que começa no *i*-ésimo símbolo, para $j = 1,...,n$.

⊗
(a) Uma substituição simples *x*/*t satisfaz o teste de ocorrência* se e somente se *x* não ocorre em *t*.
(b) Um conjunto de discórdia **D** *satisfaz o teste de ocorrência* se e somente se existem uma variável *x* e um termo *t* em **D** tais que *x* não ocorre em *t*.

Algoritmo 1: Algoritmo da Unificação

entrada: um conjunto **L** de literais
saída: - um u.m.g. β de **L**, se **L** for unificável
- 'NÃO', se **L** não for unificável

início

$\beta := \varepsilon$;
W := **L**;
D := conjunto de discórdia de W;
enquanto (número de literais de W) > 1 e **D** satisfaz o teste de ocorrência *faça*
início
selecione uma variável *x* e um termo *t* em **D** tais que *x* não ocorre em *t*;
$\beta := \beta \circ \{x/t\}$;
W := W{*x*/*t*};
D := conjunto de discórdia de W;
fim;
se (número de literais de W) = 1
então retorne β
senão retorne 'NÃO'
fim

3.1.3. O que é resolução

O sistema formal da resolução trabalha exclusivamente com cláusulas e contém apenas uma regra de inferência, chamada de regra da resolução, que gera uma nova cláusula a partir de duas outras. Dado um conjunto **S** de cláusulas e uma cláusula C, uma dedução de C a partir de **S** neste sistema formal consiste de uma seqüência de cláusulas terminando em C e gerada aplicando-se repetidamente a regra da resolução. Uma

refutação a partir de **S** é uma dedução da cláusula vazia a partir de **S**. A regra da resolução é definida de tal forma que **S** é insatisfatível se e somente se existe uma refutação a partir de **S**.

Convém recordar alguns pontos antes de prosseguir. Na definição da regra da resolução, tratar-se-á uma cláusula não-vazia "$L_1 \ldots L_n$" como o conjunto finito $\{L_1, \ldots, L_n\}$ e a cláusula vazia " " como o conjunto vazio. Assim, utilizar-se-á as operações usuais de teoria dos conjuntos para definir novas cláusulas a partir de outras. Por exemplo, se "L M N" e "N P" são cláusulas, a expressão "(L M N) $\cup$ (N P)" denota a cláusula "L M N P" (a ordem dos literais no resultado é irrelevante em face da semântica das cláusulas).

Uma cláusula A é uma *instância* de B se e somente se existir uma substituição $\beta = \{x_1/t_1, \ldots, x_n/t_n\}$ de variáveis por termos tal que A é obtida substituindo-se simultaneamente x_i por t_i em B, para $i = 1,\ldots,n$. Usar-se-á $B\beta$ para denotar o resultado da substituição.

Exemplo: Seja a cláusula B = P(x) Q(x,y). Seja uma substituição β = {x/a,y/f(b)}. A instanciação de B por β, denotada por $B\beta$, é a cláusula instância A = P(a) Q(a,f(b)).

A regra da resolução combina:

- uma adaptação para cláusulas da regra Modus Ponens (regra R1)
- um processo de "unificação" de literais de duas cláusulas (regra R2)
- um processo de "unificação" de literais de uma mesma cláusula (regra da resolução RE).

Para o primeiro passo recorde que a regra Modus Ponens ditava que de P e de P $\rightarrow$ Q, pode-se derivar Q ou, equivalentemente, de P e de $\neg$P $\vee$ Q pode-se derivar Q. Uma adaptação imediata de Modus Ponens para cláusulas seria então:

Regra R1: se A' possui um literal L e A" possui um literal $\neg$L, derive A = (A' - L) $\cup$ (A" - $\neg$L).

Ou seja, A é uma lista de literais contendo, em qualquer ordem, os literais em A' e A", exceto L e $\neg$L.

Exemplo: Seja o seguinte conjunto de cláusulas:

1. P(x) $\neg$Q(y)
2. Q(y) R(z)

dá para obter, usando a regra R1, a cláusula

3. P(x) R(z)

Agora, imagine o seguinte conjunto de cláusulas, parecido com o anterior:

1. P(x) $\neg$Q(y)
2. Q(w) R(z)

não é possível mais obter a cláusula 3, pois as variáveis são diferentes. A regra R2, vai resolver este problema.

Regra R2: se A' possui um literal L' e A" possui um literal $\neg$L" e existe uma substituição β tal que L'β = L"β, derive A = (A'β - L'β) $\cup$ (A"β - $\neg$L"β)

Ou seja, A é uma lista de literais formada tomando-se, em qualquer ordem, os literais em A' e A", exceto L' e $\neg$L", e aplicando-se a substituição β a todos os literais restantes.

Exemplo: Retomando o conjunto acima

1. P(x) $\neg$Q(y)
2. Q(w) R(z)

existe uma substituição $\beta = \{ w/y \}$, que aplicada às duas cláusulas, resulta no seguinte

1. $P(x)\ \neg Q(y)$
2. $Q(y)\ R(z)$

que obviamente produz a cláusula abaixo

3. $P(x)\ R(z)$

O processo de tornar idênticos os literais em um cláusula C através de uma substituição de variáveis por termos é chamado de *unificação* e a substituição é chamada de um *unificador* de C. Um *unificador mais geral* é aquele que, intuitivamente, especifica as substituições mais simples possíveis. O processo de unificação deverá então utilizar sempre um unificador mais geral para não bloquear outras unificações. Por exemplo, $\beta = \{x/f(a),y/a\}$ e $\theta = \{x/f(y)\}$ unificam $C = P(x)\ P(f(y))$ pois $C\beta = P(f(a))$ e $C\theta = P(f(y))$. Porém θ é um unificador mais geral do que β. A substituição θ é então preferível a β pois, por exemplo, o literal "P(f(b))" é unificável com o literal em $C\theta$, mas não é unificável com o literal em $C\beta$.

Em geral, diz-se que uma cláusula B é um *fator* de uma cláusula A se e somente se existe um conjunto **L** de literais de A e existe um unificador mais geral β para **L** tal que $B = A\beta$. Note que uma cláusula A é um fator dela mesma. O processo de obter fatores de cláusulas é chamado de *fatoração*.

Regra RE: se B' e B" são fatores de cláusulas A' e A" tais que B' possui um literal L' e B" um literal $\neg$ L" e existe um unificador mais geral β para L' e L", derive $A = (B'\beta - L'\beta) \cup (B"\beta - \neg L"\beta)$

Neste caso, diz-se que a cláusula A é um *resolvente* de A' e A", que são as *cláusulas pais*.

Exemplo: Seja o seguinte conjunto de cláusulas:

1. $P(x)\ Q(z)$
2. $\neg R(y)\ P(t)\ \neg R(w)$
3. $R(v)\ \neg Q(u)$

Se usarmos a regra R1, não obteremos nada, pois esta regra não permite substituições. Se usarmos a regra R2, obteremos as seguintes cláusulas:

4. $P(t)\ \neg R(w)\ \neg Q(u)$ R2: 2,3 $\beta = \{v/y\}$
5. $P(x)\ P(t)\ \neg R(w)$ R2: 1,4 $\beta = \{z/u\}$

e assim por diante. Mas se usarmos a Regra da Resolução (RE), podemos simplificar a cláusula 2, fatorando-a:

2'. $\neg R(y)\ P(t)$ fator de 2, com $\beta = \{w/y\}$

e aí podemos continuar aplicando RE:

4. $P(t)\ \neg Q(u)$ RE: 2',3 $\beta = \{y/v\}$
5. $P(x)$ RE: 1, 4 $\beta = \{u/z\}$ e fator de $P(x)\ P(t)$

Em resumo, o sistema formal de resolução trabalha apenas com cláusulas, que são objetos com uma sintaxe bem simples, e possui apenas uma regra de inferência, a regra da resolução. Um procedimento de *refutação baseado em resolução* é então um procedimento que, dado um conjunto qualquer de cláusulas **S**, procura sistematicamente derivar a cláusula vazia utilizando apenas a regra da resolução e tendo como ponto de partida as cláusulas em **S**. A construção de tais procedimentos requer, porém, métodos especiais para tentar contornar a explosão combinatorial gerada pela liberdade de escolha de cláusulas, fatores e literais.

3.1.4. O Sistema Formal da Resolução

⊗ Uma *renomeação* para B em presença de A é uma renomeação de variáveis β tal que A e Bβ não possuem variáveis em comum. Recorde que se L é um literal da forma P ou da forma ¬P, então P é o átomo de L, denotado por |L|.

⊗ Uma cláusula A é um *fator* de uma cláusula A' se e somente se existe um conjunto **L'** de literais de A' e um u.m.g. β de **L'** tais que A = A'β.

⊗ Uma cláusula A é um *resolvente binário* de cláusulas A' e A" se e somente se existem literais L' e L" de A' e A", respectivamente, e uma substituição θ tais que

i. L' e L" têm sinais opostos e θ é um u.m.g. de {|L'|,|L"|}
ii. A = (A'θ - L'θ) ∪ (A"θ - L"θ)

Diz-se ainda que L' e L" são os *literais resolvidos* e que θ é a *substituição de resolução*.

Portanto, por convenção, o resolvente A é formado:

- eliminando L'θ de A'θ e L"θ de A"θ;
- listando os literais restantes de Aθ e A"θ em qualquer ordem;
- eliminando os literais repetidos.

⊗ Sejam A' e A" cláusulas e β uma renomeação para A" em presença de A'. Uma cláusula A é um *resolvente* de A' e A" se e somente se A é um resolvente binário de fatores de A' e A"β.

⊗ O *sistema formal da resolução, RE*, consiste de:

Classe de Linguagens: linguagens de cláusulas
Axiomas: nenhum
Regra de Inferência: Regra da Resolução (RE)
RE: se A' e A" são cláusulas e A é um resolvente de A' e A", então derive A de A' e A".

⊗ Seja **S** um conjunto de cláusulas e C uma cláusula.

(a) Uma *dedução* de C a partir de **S** no sistema formal da resolução ou, simplesmente, uma *R-dedução* de C a partir de S, é uma seqüência $\mathbf{D} = (D_1, ..., D_n)$ de cláusulas tal que:

i. $D_n = C$
ii. para todo $i \in [1,n]$, D_i pertence a **S** ou D_i é um resolvente de D_j e D_k, para algum $j, k < i$.

Para cada $i \in [1,n]$, D_i é uma *cláusula de entrada* em **D** se e somente se D_i pertence a **S**; caso contrário, D_i é uma *cláusula derivada*.

(b) Uma *refutação* a partir de **S** no sistema formal da resolução ou, simplesmente, uma *R-refutação* a partir de **S**, é uma R-dedução de a partir de **S**.

3.2. Refutação por Resolução

3.2.1. Introdução

No problema da prova de teorema tem-se um conjunto de fórmulas **S**, a partir das quais deseja-se provar alguma fórmula meta, W. Os sistemas baseados em resolução produzem provas por contradição, ou *refutação*. Em uma refutação por resolução, primeiro nega-se a fórmula meta, e então adiciona-se a negação ao conjunto **S**.

Este conjunto expandido é então convertido a um conjunto de cláusulas, e usa-se resolução para derivar uma contradição, representada pela cláusula vazia, .

Um simples argumento pode ser dado para justificar o processo de prova por refutação. Suponha uma fórmula W, que segue logicamente de um conjunto de fórmulas **S**; então, por definição, toda interpretação que satisfaz **S** pode satisfazer ¬W, e, portanto, nenhuma interpretação pode satisfazer a união de **S** e {¬W}. Portanto, se W segue logicamente de **S**, o conjunto **S** ∪ {¬W} é insatisfatível.

Se a resolução é aplicada repetidamente a um conjunto de cláusulas insatisfatíveis, eventualmente a cláusula vazia, , será produzida. Portanto, se W segue logicamente de **S**, então a resolução eventualmente produzirá a cláusula vazia a partir da representação da cláusula **S** ∪ {¬W}. Por outro lado, se a cláusula vazia é produzida a partir da representação de cláusula **S** ∪ {¬W}, então W segue logicamente de **S**.

Considere um exemplo simples. Observe as seguintes frases:

(1) Qualquer um que possa ler é alfabetizado.
$\forall x\,(L(x) \rightarrow A(x))$
(2) Os golfinhos não são alfabetizados.
$\forall x\,(G(x) \rightarrow \neg A(x))$
(3) Alguns golfinhos são inteligentes.
$\exists x\,(G(x) \wedge I(x))$

A partir destes quer-se provar a frase:

(4) Alguns que são inteligentes não podem ler.
$\exists x\,(I(x) \wedge \neg L(x))$

O conjunto de cláusulas que correspondem às frases 1 a 3 é:

(1) $\neg L(x)\ A(x)$
(2) $\neg G(y)\ \neg A(y)$
(3a) G(a)
(3b) I(a)

onde *a* é a constante de Skolem. A negação do teorema a ser provado, convertido a forma de cláusula, é:

(4') $\neg I(z)\ L(z)$.

Provar este teorema através da refutação por resolução envolve gerar resolventes a partir do conjunto de cláusulas 1-3 e 4', adicionando estes resolventes ao conjunto, e continuando até que a cláusula vazia seja produzida. Uma prova possível (existe mais de uma) produz a seguinte seqüência de resolventes:

(5) L(a) resolvente de 3b e 4'
(6) A(a) resolvente de 5 e 1
(7) ¬G(a) resolvente de 6 e 2
(8) resolvente de 7 e 3a.

3.2.2. Sistemas de Produção para refutação por resolução

Suponha um conjunto **S** de cláusulas chamado de *conjunto base*. O algoritmo básico para um sistema de produção de refutação por resolução pode ser escrito como:

Procedimento *RESOLUÇÃO*

1 *CLÁUSULAS* ← **S**
2 *até que* seja um membro de *CLÁUSULAS*, *faça*:

3 *início*
4 selecione duas cláusulas distintas C_i e C_j em *CLÁUSULAS*
5 calcule um resolvente, R_{ij} de C_i e C_j
6 *CLÁUSULAS* ← o conjunto produzido adicionando R_{ij} a *CLÁUSULAS*
7 *fim*

3.2.3. Estratégias de Controle para Métodos de Resolução

As decisões sobre quais cláusulas em *CLÁUSULAS* resolver (comando 4) e qual resolução destas cláusulas realizar (comando 5) são tomadas através da estratégia de controle.

É útil para a estratégia de controle usar uma estrutura chamada de *grafo de derivação*. Os nós neste grafo são rotulados pelas cláusulas; inicialmente, existe um nó para toda cláusula no conjunto base. Quando duas cláusulas C_i e C_j produzem um resolvente R_{ij} , cria-se um novo nó, *descendente*, rotulado R_{ij} , ligado com os nós pais C_i e C_j.

Uma refutação por resolução pode ser representada como uma *árvore de refutação* (dentro do grafo de derivação) tendo um nó raiz rotulado por .

A estratégia de controle busca por uma refutação crescendo o *grafo* de derivação até que uma *árvore* seja produzida com um nó raiz rotulado pela cláusula vazia, . Uma estratégia de controle para um sistema de refutação é *completa* se seu uso resulta num procedimento que achará uma contradição (eventualmente) onde existir. (A completude de uma *estratégia* não deve ser confundida com a completude lógica de uma regra de inferência).

3.2.3.1. Estratégias para Provar uma Meta através da Refutação

a) A ESTRATÉGIA DE BUSCA EM LARGURA

Na estratégia de busca em largura, todos os resolventes de primeiro nível são calculados primeiro, depois os resolventes de segundo nível, e assim por diante. (Um *resolvente de primeiro nível* está entre as cláusulas do conjunto base; um *resolvente do i-ésimo nível* é aquele cujos pais são resolventes do (*i* - 1)-ésimo nível.) A estratégia de busca em largura é completa, mas é muito ineficiente.

b) A ESTRATÉGIA DO CONJUNTO DE SUPORTE

Uma refutação por conjunto de suporte é aquela na qual no mínimo um pai para cada resolvente é selecionado entre as cláusulas resultantes da negação da fórmula meta ou dos seus descendentes (o *conjunto de suporte*). A estratégia precisa garantir a busca de todos as refutações por conjunto de suporte possíveis (na forma por largura). Além de completa, a estratégia do conjunto de suporte é mais eficiente que a busca em largura.

c) A ESTRATÉGIA POR PREFERÊNCIA UNITÁRIA

A estratégia por preferência unitária é uma modificação da estratégia por conjunto de suporte na qual, ao invés de preencher cada nível na forma por largura, tenta-se selecionar uma cláusula de um único literal (chamado de *unidade*) para ser um pai numa resolução. Cada vez que as unidades são usadas na resolução, os resolventes têm menos literais do que seus outros pais. Este processo ajuda a dirigir a busca para produzir a cláusula vazia e, então, tipicamente, aumentar a eficiência. Mas não é completa.

3.2.3.2. Estratégias para Provar que um Conjunto de Cláusulas é Insatisfatível

d) A ESTRATÉGIA POR FORMA DE ENTRADA LINEAR

Uma refutação por forma de entrada linear é aquela na qual cada resolvente tem no mínimo um pai pertencente ao conjunto base. Esta estratégia não é completa, ou seja, existem casos nos quais uma refutação existe mas uma refutação por forma de entrada linear não.

e) A ESTRATÉGIA POR FORMA "ANCESTRAL FILTRADA"

Uma refutação por forma "ancestral filtrada" é aquela onde cada resolvente tem um pai que está no conjunto base ou que é um ancestral do outro pai. Portanto, a forma "ancestral filtrada" é muito parecida com a forma linear. É uma estratégia completa.

3.2.4. Estratégias de Simplificação

Algumas vezes um conjunto de cláusulas pode ser simplificado pela eliminação de certas cláusulas ou pela eliminação de certos literais dentro das cláusulas. Estas simplificações são tais que o conjunto de cláusulas simplificado é insatisfatível se e somente se o conjunto original for insatisfatível. Portanto, o emprego destas estratégias de simplificação ajuda a reduzir a taxa de crescimento de novas cláusulas.

a) ELIMINAÇÃO DE TAUTOLOGIAS

Qualquer cláusula contendo um literal e sua negação (chama-se tal cláusula uma *tautologia*) pode ser eliminada, desde que qualquer conjunto insatisfatível contendo uma tautologia ainda seja insatisfatível depois de sua remoção. e vice-versa.

b) INCORPORAÇÃO PROCEDIMENTAL

Algumas vezes é possível e mais conveniente *calcular* os valores verdade de literais do que incluir estes literais, ou suas negações, no conjunto base. Tipicamente, os cálculos são realizados para instâncias concretas. Uma *instância concreta* é uma instância de uma expressão onde não ocorrem variáveis.

Mas o que significa realmente "calcular" uma expressão. As expressões do cálculo de predicados são construções lingüísticas que denotam valores verdade, elementos, funções ou relações num domínio. Tais expressões podem ser interpretadas com referência a um modelo que associa entidades lingüísticas com entidades de domínio apropriadas. O resultado final é que os valores V ou F tornam-se associados com sentenças na linguagem.

c) ELIMINAÇÃO POR SUBJUGAÇÃO

Por definição, uma cláusula A_i *subjuga* uma cláusula B_i se existe uma substituição β tal que A_i β é um subconjunto de B_i . Como exemplos:

P(*x*) subjuga P(*y*) Q(*z*), para β = { *x* / *y* }
P(*x*) subjuga P(a), para β = { *x* / a }
P(*x*) subjuga P(a) Q(*z*), para β = { *x* / a }
P(*x*) Q(a) subjuga P(f(a)) Q(a) R(*y*), para β = { *x* / f(a) }

Uma cláusula num conjunto insatisfatível que é subjugada por uma outra cláusula no conjunto pode ser eliminada sem afetar a insatisfatibilidade do resto do conjunto. A eliminação de cláusulas subjugadas por outras freqüentemente leva a reduções substanciais no número de resoluções necessárias para encontrar uma refutação.

IV. SISTEMAS DE DEDUÇÃO BASEADOS EM REGRAS

4.1. Introdução

A forma como um pedaço de conhecimento sobre um certo campo é expresso por um especialista deste campo, freqüentemente contém informação importante sobre como esse conhecimento pode ser usado da melhor forma. Suponha, por exemplo, que um matemático diga:

"Se *x* e *y* são ambos maiores que zero, o produto de *x* e *y* também é maior que zero."

No cálculo de predicados esta sentença ficaria:

$\forall x \forall y\ ((M(x,0) \wedge M(y,0)) \rightarrow M(vezes(x,y),0))$.

Entretanto, pode-se usar a seguinte fórmula equivalente:

$\forall x \forall y\ ((M(x,0) \wedge \neg M(vezes(x,y),0)) \rightarrow \neg M(y,0))$.

O conteúdo lógico da sentença do matemático é independente das muitas formas equivalentes do cálculo de predicados que poderiam representá-la. Mas, na forma que as sentenças do Português são construídas, freqüentemente contêm informação de controle extra-lógica ou heurística. No exemplo acima, a sentença parece indicar que se está habituado ao fato de que se *x* e *y* são individualmente maiores que zero, então é óbvio provar que *x* multiplicado por *y* é maior que zero.

Muito do conhecimento usado por sistemas de IA é diretamente representável por expressões gerais de implicação. Veja as seguintes sentenças:

(1) Todos os vertebrados são animais.
$\forall x$ (Vertebrado (x) $\rightarrow$ Animal (x))

(2) Todos do departamento de computação acima de 30 anos são casados.
$\forall x \forall y$ ((Trabalha_em(dep_computação,x) $\wedge$ Idade (x,y) $\wedge$ M(y,30)) $\rightarrow$ Casado (x))

(3) Existe um cubo acima de todo cilindro vermelho.
$\forall x$ ((Cilindro (x) $\wedge$ Vermelho (x)) $\rightarrow \exists y$ (Cubo (y) $\wedge$ Acima (y,x)))

O sistema descrito aqui não converte fórmulas em cláusulas; eles as usam numa forma perto da sua forma original dada. As fórmulas que representam conhecimento de asserção sobre o problema são separadas em duas categorias: *regras* e *fatos*. As regras consistem das asserções dadas na forma implicacional. Tipicamente, elas expressam conhecimento *geral* sobre uma área particular e são usadas como regras de produção. Os fatos são as asserções que não são expressas como implicações. Tipicamente, eles representam conhecimento *específico* relevante a um caso particular. A tarefa dos sistemas de produção é provar uma *fórmula meta* a partir destes fatos e regras.

Em sistemas *forward* (progressivo ou "data-driven"), as implicações usadas como regras-F operam numa base de dados global de fatos até que uma condição de terminação envolvendo a fórmula meta seja alcançada. Em sistemas *backward* (regressivo ou "goal-driven") as implicações usadas como regras-B operam numa base de dados global de metas até que uma condição de terminação envolvendo os fatos seja alcançada. Combinar operação *forward* e *backward* também é possível.

Este tipo de sistema de prova de teorema é um sistema *direto* ao contrário do sistema de refutação. Um sistema direto não é necessariamente mais eficiente que um sistema de refutação, mas sua operação parece ser intuitivamente mais fácil para as pessoas entenderem.

Sistemas deste tipo são freqüentemente chamados de *sistemas de dedução baseados em regras*, para enfatizar a importância de se usar regras para fazer deduções. A pesquisa em IA tem produzido muitas aplicações de sistemas baseados em regras.

4.2. Um Sistema de Dedução *Forward*

4.2.1. A forma AND/OR para Expressões de Fatos

O sistema *forward* tem como sua base de dados global inicial uma representação para o conjunto de fatos dado. Em particular, não se pretende converter estes fatos em forma de cláusulas. Os fatos são representados como fórmulas do cálculo de predicados que foram transformadas em formas livres de implicações chamadas *formas AND/OR*. Para converter uma fórmula na forma AND/OR, os símbolos "$\rightarrow$" (se existirem) são eliminados, usando a equivalência de $(W1 \rightarrow W2)$ e $(\neg W1 \vee W2)$. (Tipicamente, existem poucos símbolos "$\rightarrow$" entre os fatos porque as implicações são preferivelmente representadas como regras.) Depois, os símbolos de negação são movidos para dentro (usando as leis de De Morgan) até que seus escopos incluam, no máximo, um único predicado. A expressão resultante é então Skolemizada e prenexada; as variáveis dentro dos escopos dos quantificadores universais são padronizadas através da renomeação, as variáveis quantificadas existencialmente são substituídas por funções de Skolem, e os quantificadores universais são eliminados. Qualquer variável restante é assumida ter quantificação universal. Ou seja, na verdade é aplicado o algoritmo da representação clausal até o passo imediatamente anterior a obtenção da forma normal conjuntiva, complementando com a eliminação dos quantificadores universais.

Por exemplo, a expressão fato:

$$\exists u \forall v\, (Q(v,u) \wedge \neg((R(v) \vee P(v)) \wedge S(u,v)))$$

é convertida para

$$Q(v,a) \wedge ((\neg R(v) \wedge \neg P(v)) \vee \neg S(a,v)))$$

As variáveis podem ser renomeadas de tal forma que a mesma variável não ocorra em conjunções diferentes (principais) da expressão fato. A renomeação de variáveis neste exemplo leva à expressão:

$$Q(w,a) \wedge ((\neg R(v) \wedge \neg P(v)) \vee \neg S(a,v)).$$

Uma expressão na forma AND/OR consiste de subexpressões de literais conectados por símbolos "$\wedge$" e "$\vee$". Note que uma expressão na forma AND/OR não está na forma de cláusula. Está muito mais perto da expressão original.

4.3. Um Sistema de Dedução *backward*

Uma propriedade importante da lógica é a dualidade entre asserções e metas em sistemas de prova de teoremas. Já foi visto uma instância deste princípio de dualidade nos sistemas de refutação por resolução. Lá a fórmula meta era negada, convertida na forma de cláusula, e adicionada às asserções em forma de cláusulas também. A dualidade entre asserções e metas permite que a meta negada seja tratada como se fosse uma asserção. Os sistemas de refutação por resolução aplicam resolução ao conjunto de cláusulas combinadas até que a cláusula vazia (denotando F) seja produzida.

Pode-se também descrever um sistema de resolução dual que opera nas expressões metas. Para preparar as fórmulas para tal sistema, deve-se primeiro negar a fórmula representando as asserções, converter esta fórmula negada ao dual da fórmula de cláusula (uma disjunção de conjunções de literais), e adicionar estas cláusulas a forma de cláusula dual da fórmula meta. O sistema deve então aplicar uma versão dual da resolução até que a cláusula vazia (agora denotando V) seja produzida.

Pode-se também imaginar sistemas mistos nos quais três formas diferentes de resolução são usadas, a resolução entre asserções, a resolução entre expressões metas e a resolução entre uma asserção e uma meta. O sistema *forward* descrito no último item deveria ser apontado como um destes sistemas mistos porque ele envolve o casamento de um literal fato no grafo AND/OR com um literal meta. O sistema de produção *backward*, descrito aqui, é também um sistema misto que é, de alguma forma, dual ao sistema *forward*. Sua operação envolve os mesmos tipos de representações e mecanismos que são usados no sistema *forward*.

4.3.1. Expressões Metas na Forma AND/OR

O sistema *backward* é capaz de tratar de expressões metas de forma arbitrária. Primeiro, converte-se a fórmula meta na forma AND/OR pelo mesmo tipo de processo usado para converter uma expressão fato. Elimina-se os símbolos →, move-se os símbolos de negação para dentro, Skolemiza-se as variáveis *universais*, e elimina-se os quantificadores existenciais. As variáveis restantes na forma AND/OR de uma expressão meta têm assumida a quantificação existencial.

Por exemplo, a expressão meta:

$$\exists y \forall x\, (P(x) \rightarrow (Q(x,y) \wedge \neg(R(x) \wedge S(y))))$$

é convertida para

$$\neg P(f(y)) \vee (Q(f(y),y) \wedge (\neg R(f(y)) \vee \neg S(y))),$$

onde f(y) é uma função de Skolem.

A padronização das variáveis nas disjunções (principais) da meta leva a:

$$\neg P(f(z)) \vee (Q(f(y),y) \wedge (\neg R(f(y)) \vee \neg S(y))).$$

(Note que a variável y não pode ser renomeada *dentro* da subexpressão disjuntiva.)

A linguagem Prolog trabalha no estilo *backward*. Por exemplo, seja o seguinte programa:

```
p(X,Y) :- q(X,Y).
q(a,b).
```

E se coloque a seguinte questão:

```
?- p(a,b).
```

Prolog faz as instanciações X = a e Y = b, e dispara a primeira regra, ou seja, troca p(a,b) por q(a,b). (Na verdade, como p(X,Y) :- q(X,Y) corresponde à implicação da lógica q(X, Y) → p(X,Y), então o Prolog está trocando o conseqüente da implicação p(a,b) pelo seu antecedente q(a,b), ou seja, estratégia *backward*).

4.4. Uma Combinação de Sistemas *forward* e *backward*

Ambos os sistemas de dedução baseados em regras, *forward* e *backward*, têm limitações. O sistema *backward* pode manipular expressões metas de forma arbitrária mas está restrito a expressões fatos consistindo de conjunções de literais. O sistema *forward* pode manipular expressões fatos de forma arbitrária mas está restrito a expressões metas consistindo de disjunções de literais. Pode-se combinar os dois sistemas e tirar vantagens de cada um sem as limitações de ambos?

Na verdade, quatro fatores influenciam a razão de se escolher raciocinar *forward* ou *backward* (Rich e Knight, 1994):

1. Existem mais estados iniciais ou metas? É preferível mover de um conjunto de estados menor para um maior (e portanto mais fácil de achar).
2. Em qual direção o fator de ramificação é maior? (fator de ramificação é o número médio de nós que podem ser alcançados diretamente de um único nó.) É preferível mover na direção com o fator de ramificação menor.
3. O programa terá que justificar seu processo de raciocínio para o usuário? Se tiver, é importante mover na direção que corresponde à forma como o usuário está pensando.
4. Que tipo de evento irá iniciar um episódio de resolução de problema? Se for a chegada de um novo fato, o raciocínio *forward* faz sentido. Se for desejada a resposta de uma pergunta, o raciocínio *backward* é mais natural.

V. REDES NEURAIS ARTIFICIAIS

5.1. Introdução

A evolução natural deu ao cérebro humano muitas características desejáveis que não estão presentes na máquina de von Neumann (os computadores atuais) tais como (Jain e Mao, 1996):

- Paralelismo massivo
- Representação e computação distribuídas
- Habilidade de aprendizado
- Habilidade de generalização
- Adaptabilidade
- Processamento de informação contextual inerente
- Tolerância a falhas
- Baixo consumo de energia

É desejável que os dispositivos baseados nas redes neurais biológicas possuam algumas destas características. Veja na tabela a seguir (de Jain e Mao, 1996), a comparação entre o computador de von Neumann e o sistema neural biológico.

	Computador de von Neumann	**Sistema neural biológico**
Processador	Complexo Alta velocidade Um ou poucos	Simples Baixa velocidade Um grande número
Memória	Separado do processador Localizado Não-endereçável pelo conteúdo	Integrada com o processador Distribuída Endereçável pelo conteúdo
Computação	Centralizada Seqüencial Programas armazenados	Distribuída Paralela Auto-aprendizado
Confiabilidade	Muito vulnerável	Robusta
Especialidade	Manipulações numéricas e simbólicas	Problemas perceptuais
Ambiente operacional	Bem definido, bem restrito	Pobremente definido, irrestrito

Cérebros e computadores digitais realizam tarefas bem diferentes e têm propriedades diferentes. Veja a tabela abaixo (de Russell e Norvig, 1995) que mostra uma comparação entre cérebros e computadores digitais (de 1994):

	Computador de 1994	Cérebro humano
Unidades computacionais	1 CPU, 10^5 portas	10^{11} neurônios
Unidades de armazenamento	RAM de 10^9 bits, disco de 10^{10} bits	10^{11} neurônios, 10^{14} sinapses
Tempo de ciclo	10^{-8} seg	10^{-3} seg
Bandwidth	10^9 bits/seg	10^{14} bits/seg
Atualizações de neurônio/seg	10^5	10^{14}

5.2. O perceptron

O primeiro modelo matemático do neurônio foi o modelo proposto por McCulloch e Pitts em 1943. Mais tarde, Rosenblatt (1957) criou o modelo do perceptron. Um perceptron modela um neurônio tomando uma soma ponderada de suas entradas e enviando a saída 1 se esta soma é maior que um determinado limiar (senão, envia 0). Veja a figura 1.

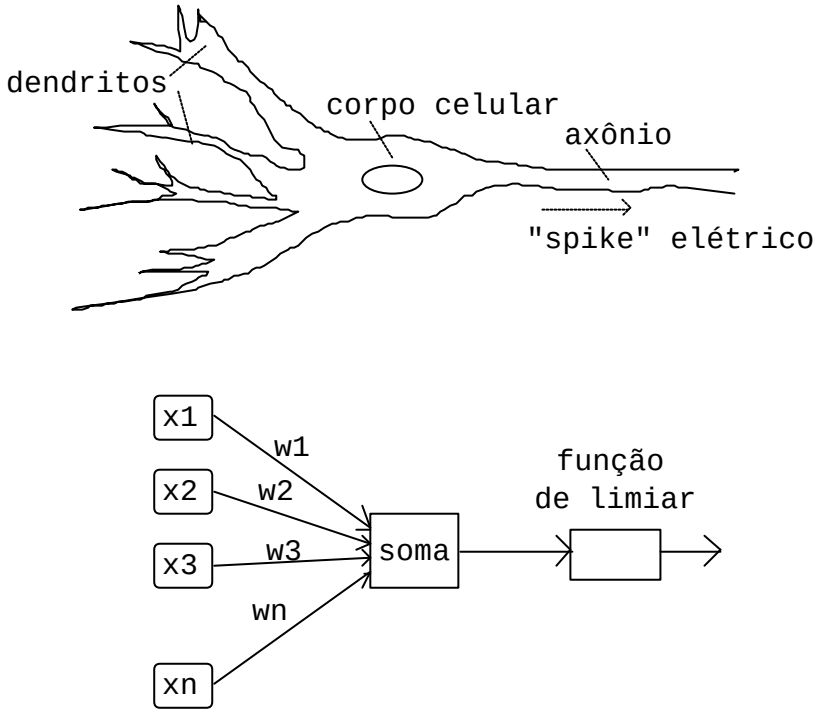

Figura 1. Um neurônio e um perceptron (Rich e Knight, 1994).

5.3. Algoritmos conexionistas

Uma vez identificado o problema que se queira solucionar através da abordagem conexionista, deve-se construir a rede neural. Ou seja, montar a arquitetura da rede: para uma rede de três camadas, quantos neurônios eu devo ter na entrada da rede (que corresponde, normalmente, ao número de bits que representa o meu padrão), quantos eu devo ter na saída (que corresponde, normalmente, a quantidade de bits do meu padrão de saída) e, o mais difícil, o número de neurônios na camada escondida. Os neurônios da camada escondida normalmente não são "calculados" e devem ser tentados empiricamente.

Depois de construída a rede neural artificial, deve-se escolher um *algoritmo conexionista* para "treinar" a rede (fase de aprendizado). O treinamento da rede normalmente é demorado, pois requer muitos "ciclos", ou

seja, deve-se mostrar a rede várias vezes, tudo que se deseja que ela aprenda. Depois do treinamento, a rede neural deve ser capaz de, numa única propagação (único ciclo) reconhecer o padrão no qual ela foi ensinada (fase de reconhecimento).

Os algoritmos de redes neurais em geral se dividem em dois tipos básicos: os algoritmos *supervisionados*, ou seja, quando a saída desejada da rede durante o treinamento é fornecida para comparação, e os *não-supervisionados*, quando a rede se conduz por si só, ou seja, não há um supervisor que verifique as suas saídas.

Entre os algoritmos supervisionados mais conhecidos está o algoritmo *backpropagation* (McClelland e Rumelhart, 1986). Neste algoritmo, a cada ciclo o padrão de entrada é propagado pela rede e na saída ele é comparado com a saída desejada (supervisor). Caso haja erros, os mesmos são corrigidos gradativamente, através das mudanças de pesos dos neurônios que se conectam à saída errada (propagação de volta dos erros).

Entre os algoritmos não-supervisionados mais representativos está o *competitive learning*, ou aprendizado competitivo, já discutido anteriormente.

5.3.1. Redes Multicamadas

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 2. 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.

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, por exemplo.

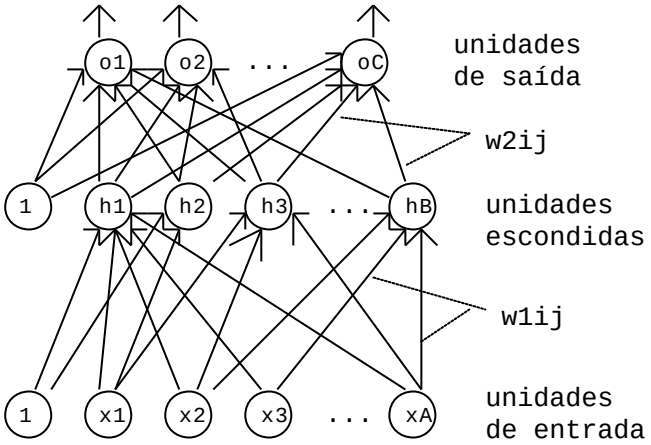

Figura 2. Uma rede multicamada (Rich e Knight, 1994).

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 ponderadas 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 algoritmo 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.

5.4. Redes Neurais Baseadas em Conhecimento

A rede neural trabalha muito bem ao resolver certos tipos de problemas. A maior crítica que se faz aos sistemas conexionistas é o fato de que sabe-se que funciona mas não se sabe como. A representação interna dos pesos de uma rede neural é uma incógnita para os pesquisadores. Mas este tipo de crítica está sendo respondida gradativamente. Já há alguns anos começaram as pesquisas em Redes Neurais Baseadas em Conhecimento, ou seja, redes neurais onde um conhecimento inicial é representado. A rede não começa mais com pesos sinápticos aleatórios e sim com um conjunto de pesos que refletem regras de produção. Na verdade trata-se de uma mistura das abordagens simbólica e conexionista (a chamada abordagem *híbrida*).

Numa rede baseada em conhecimento, dada uma teoria simbólica, cria-se a partir das regras desta teoria, uma arquitetura de rede neural. Ou seja, de uma regra $A \rightarrow B$, cria-se uma conexão entre um neurônio que representa o conceito A e outro neurônio que representa o conceito B. Para uma regra $(A \wedge B) \rightarrow C$, tem-se

dois neurônios de entrada A e B, um neurônio na camada escondida faz o papel da conjunção e um neurônio na camada de saída representa o conceito C. Basta ligá-los através de pesos de conexão grandes e tem-se uma rede que representa esta regra. A função da rede neural é revisar esta teoria. A teoria inicial (regras) está representada na rede. A rede começa a aprender. No final do aprendizado pode-se extrair de volta as regras da rede. A teoria representada pelas regras foi revisada pelo aprendizado. Fu (1993), Setiono e Liu (1996) e outros apresentam sistemas neurais baseados em conhecimento.

Bibliografia

- Casanova, M. A., Giorno, F. A. C. e Furtado, A. L. (1987). *Programação em Lógica e a Linguagem Prolog*. Editora Edgard Blücher Ltda.
- Fu, L. M. (1993). Knowledge-Based Connectionism for Revising Domain Theories. *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, no. 1, Jan/Feb, 173-182.
- Jain, A. K. and Mao, J. (1996). Artificial Neural Networks; A Tutorial. *Computer*, vol. 29, no. 3, March, 31-44.
- McClelland, J. L. and Rumelhart, D. E. and the PDP Research Group. (1986). *Parallel Distributed Processing - Explorations in the Microstructure of Cognition - Volume 2: Psychological and Biological Models*. A Bradford Book, The MIT Press.
- McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115-137. *Apud* Russell and Norvig (1995).
- Nilsson, N. J. (1982). *Principles of Artificial Intelligence*. Springer-Verlag.
- Rich. E. e Knight, K. (1994). *Inteligência Artificial* - 2ª. edição. Makron Books.
- Rosa, J. L. G. (1999), *Um Sistema Híbrido Simbólico-conexionista para o Processamento de Papéis Temáticos.* Tese de Doutorado. Departamento de Lingüística. Instituto de Estudos da Linguagem. Universidade Estadual de Campinas – DL-IEL-Unicamp. Junho.
- Rosa, J. L. G. (1993). *Redes Neurais e Lógica Formal em Processamento de Linguagem Natural*. Dissertação de Mestrado. Departamento de Engenharia de Computação e Automação Industrial - Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas – DCA-FEEC-Unicamp. Setembro.
- Rosa, J. L. G. and Françozo, E. (2000), Linguistic Relations Encoding in a Symbolic-Connectionist Hybrid Natural Language Processor, in Proceedings of the International Joint Conference SBIA/IBERAMIA'2000, Lecture Notes in Artificial Intelligence, Springer-Verlag, edited by Maria Carolina Monard, Atibaia, São Paulo, Brazil, November 19-22 (accepted for publication).
- Rosa, J. L. G. and Françozo, E. (1999), Hybrid Thematic Role Processor: Symbolic Linguistic Relations Revised by Connectionist Learning, in *Proceedings of IJCAI'99 – Sixteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, July 31-August 6, Volume 2, Morgan Kaufmann, 852-857.
- Rosa, J. L. G. and Françozo, E. (1998), A Biologically Fine-grained Artificial Neural Network: Towards a Hybrid Model, *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, May 27-30, Cancún, México, 302-305.
- Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automaton. *Report 85-460-1, Project PARA, Cornell Aeronautical Laboratory*, Ithaca, New York. *Apud* Russell and Norvig (1995).
- Russell, S. and Norvig, P. (1995). *Artificial Intelligence – A Modern Approach*. Prentice-Hall.
- Setiono, R. and Liu, H. (1996). Symbolic Representation of Neural Networks. *Computer* vol. 29, no. 3, march, 71-77.