Raciocínio Automatizado

Prof. Dr. Silvio do Lago Pereira

slago@ime.usp.br

1 Introdução

Raciocínio automatizado é uma sub-área da IA que estuda formas de simular racicínio lógico por meio de métodos computacionais. Um dos principais algoritmos para raciocínio dedutivo automatizado, denominado SLD-*resolução* [3], usa refutação e apresenta as seguintes características:

- restringe-se à uma classe de fórmulas denominada *cláusulas de Horn*;
- emprega resolução e unificação como regras de inferência;
- adota uma estratégia de busca em profundidade para controlar as inferências;
- introduz o conceito de *predicado computável*;
- introduz o conceito de *negação por falha finita*.

2 Claúsulas de Horn

Cláusulas de Horn [1] são fórmulas da forma $\phi \leftarrow \phi_1, \ldots, \phi_n$, sendo $n \geq 0$, onde o literal ϕ é uma conclusão e os literais ϕ_i são premissas. Se $n = 0$, a cláusula é denominada *fato* e se $n > 0$, a cláusula é denominada *regra*. Uma fórmula da forma $\leftarrow \phi_1, \phi_2, \ldots, \phi_n$ é denominada *consulta*. A cláusula $\leftarrow$ é denominada *vazia* e denota uma contradição.

2.1 Inferência com cláusulas de Horn

Inferências com cláusulas de Horn são realizadas sempre entre uma consulta e um fato ou entre uma consulta e uma regra, sendo que o resultado da inferência, denominado *resolvente*, é sempre uma consulta ou a clásula vazia.

- **1^0 caso**: se o fato $\alpha_0 \leftarrow$ unifica-se com o primeiro literal da consulta $\leftarrow \beta_1, \beta_2, \ldots, \beta_n$, sob o conjunto de substituições σ, então a resolvente será a consulta $\leftarrow \beta'_2, \ldots, \beta'_n$, onde β'_i é uma instância de β_i, sob a substituição σ;
- **2^0 caso**: se a conclusão da regra $\alpha_0 \leftarrow \alpha_1, \ldots, \alpha_m$ unifica-se com o primeiro literal da consulta $\leftarrow \beta_1, \beta_2, \ldots, \beta_n$, sob o conjunto de substituições σ, então a resolvente será a consulta $\leftarrow \alpha'_1, \ldots, \alpha'_m, \beta'_2, \ldots, \beta'_n$, onde α'_i é uma instância de α_i e β'_i é uma instância de β_i, sob a substituição σ.

A função $unifica(\alpha, \beta, \gamma)$ tenta unificar a cláusula α com a consulta β, resultando na nova consulta γ (que pode ser a cláusula vazia). Caso a unificação seja bem sucedida, essa função devolve `verdade`; caso contrário, devolve `falso`.

2.2 O algoritmo de busca SLD-Resolução

O algoritmo SLD-Resolução [1,2] controla a aplicação da regra de inferência, realizando uma busca em profundidade. Ao derivar a cláusula vazia, o algoritmo apresenta como solução a composição das substituições efetuadas no caminho percorrido até a cláusula vazia e sinaliza `sucesso`. Ao atingir um ponto onde a consulta não pode ser unificada com nenhuma cláusula do programa, o algoritmo sinaliza `fracasso` e tenta retroceder na busca fazendo novas escolhas.

Sejam Δ um conjunto de cláusulas de Horn (programa lógico) e β uma consulta. O algoritmo a seguir responde à consulta β com base em Δ:

SLD-Resolução(Δ, β)
1 `se` β *é a cláusula vazia* `então`
2 `exiba` *a composição das substituições efetuadas*
3 `devolva sucesso`
4 `para cada` *cláusula* $\alpha \in \Delta$ (*de cima para baixo*) `faça`
5 `se` $unifica(\alpha, \beta, \gamma)$ `e` SLD-Resolução(Δ, γ) =`sucesso então`
6 `devolva sucesso`
7 `devolva fracasso`

Para mostrar como esse algoritmo funciona, apresentamos dois exemplos.

Exemplo 1 Seja Δ o programa lógico composto pelas cláusulas a seguir:

(1) $bebe(ze, pinga) \leftarrow$
(2) $bebe(mane, agua) \leftarrow$
(3) $vivo(mane) \leftarrow$
(4) $saudavel(X) \leftarrow bebe(Y, X), vivo(Y)$

A cláusula (4) de Δ estabelece que uma bebida é saudavel se alguém que a bebe está vivo. Suponha que desejamos descobrir, de acordo com o conhecimento expresso em Δ, quais são as bebidas saudáveis. Para tanto, chamamos o algoritmo SLD-Resolução com a consulta $\leftarrow saudavel(R)$. O algoritmo começa tentando eliminar $saudavel(R)$ da consulta. Ele procura em Δ uma cláusula cuja conclusão possa ser unificada com $saudavel(R)$, encontrando a cláusula (4). Então, aplicando a substituição $\{X = R\}$ a essa cláusula, o algoritmo obtém a instância $saudavel(R) \leftarrow bebe(Y, R), vivo(Y)$ que, combinada com a consulta $\leftarrow saudavel(R)$, produz a resolvente $\leftarrow bebe(Y, R), vivo(Y)$. Em seguida, o algoritmo precisa eliminar $bebe(Y, R)$ e, para isso, pode usar as cláusulas (1) ou (2) do programa Δ:

- usando a cláusula (1) sob a substituição $\{Y = ze, R = pinga\}$, a consulta é reduzida para $\leftarrow vivo(ze)$; porém, como não há em Δ uma cláusula que possa ser resolvida com essa nova consulta, o algoritmo retrocede na sua escolha e tenta a outra alternativa;

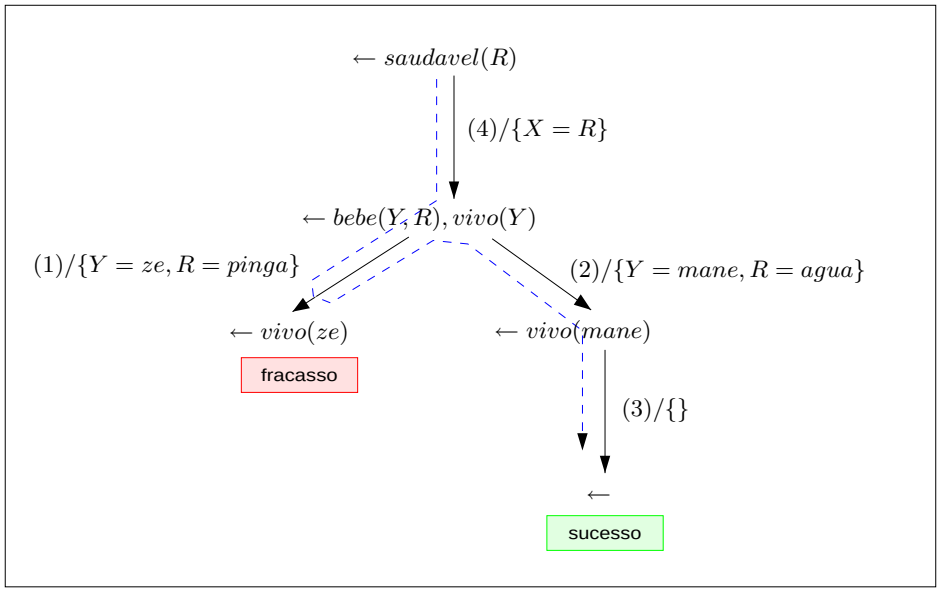

Figura 1. Árvore de refutação para a consulta ← $saudavel(R)$

– usando a cláusula (2) sob a substituição $\{Y = mane, R = agua\}$, a consulta é reduzida para ← $vivo(mane)$ que, combinada com a cláusula (3), resulta numa contradição, representada pela cláusula vazia ←.

Quando a cláusula vazia é derivada, o algoritmo apresenta o resultado das unificações realizadas no caminho que levaram a ela (ou seja, $R = agua$) e finaliza a busca. Esse processo pode ser visualizado na Figura 1, onde a linha tracejada indica o caminho percorrido pelo algoritmo na árvore de refutação para a consulta ← $saudavel(R)$. □

Exemplo 2 Seja Δ o programa lógico composto pelas cláusulas a seguir:

(1) $nasceu(ana, brasil) \leftarrow$
(2) $nasceu(yves, france) \leftarrow$
(3) $idioma(brasil, portugues) \leftarrow$
(4) $idioma(franca, frances) \leftarrow$
(5) $estudou(ana, frances) \leftarrow$
(6) $fala(A, C) \leftarrow nasceu(A, B), idioma(B, C)$
(7) $fala(D, E) \leftarrow estudou(D, E)$

Para descobrir que idiomas Ana fala, chamamos SLD-RESOLUÇÃO com a consulta ← $fala(ana, R)$, que pode ser resolvida com as cláusulas (6) e (7):

– Usando a cláusula (6), sob a substituição $\{A = ana, C = R\}$, obtemos a consulta ← $nasceu(ana, B), idioma(B, R)$. Resolvendo essa nova consulta com a

cláusula (1), sob a substituição $\{B = brasil\}$, obtemos $\leftarrow idioma(brasil, R)$. Finalmente, resolvendo essa última consulta com a cláusula (3), sob a substituição $\{R = portugues\}$, obtemos a cláusula vazia. Nesse ponto, o algoritmo apresenta a primeira resposta possível ($R = portugues$) e retrocede na busca, na tentativa de encontrar uma resposta alternativa.

– Usando a cláusula (7), sob a substituição $\{D = ana, E = R\}$, obtemos a consulta $\leftarrow estudou(ana, R)$. Resolvendo essa nova consulta com a cláusula (5), sob a substituição $\{R = frances\}$, obtemos a cláusula vazia. Nesse ponto, o algoritmo apresenta a segunda resposta possível ($R = frances$) e, como não há mais ramos em aberto na árvore de refutação, a busca termina.

Para essa consulta, o algoritmo é bem sucedido nos dois ramos explorados na árvore, apresentando $R = portugues$ e $R = frances$ como respostas. Veja como isso faz sentido: declaramos no programa lógico que uma pessoa fala um idioma se ela nasce num país onde se fala esse idioma ou se ela estuda esse idioma. Então, como Ana nasceu no Brasil, o algoritmo conclui que Ana fala português. Além disso, como Ana estudou francês, o algoritmo também conclui que Ana fala francês. Esse processo pode ser visto na Figura 2, onde a linha tracejada indica o caminho percorrido pelo algoritmo na árvore de refutação para a consulta $\leftarrow fala(ana, R)$.

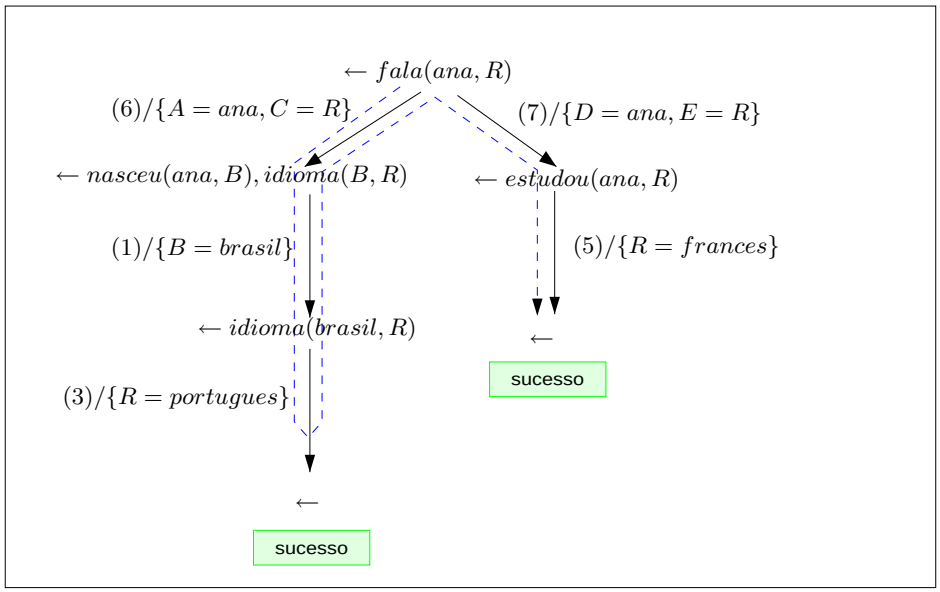

Figura 2. Árvore de refutação para a consulta $\leftarrow fala(ana, R)$

O funcionamento de SLD-RESOLUÇÃO para a consulta $\leftarrow fala(yves, R)$ é apresentado na Figura 3. Como Yves não estudou nenhum idioma, é claro que ele fala apenas francês, que é a sua língua nativa. Portanto, o algoritmo encontra apenas uma resposta para essa consulta ($R = frances$). □

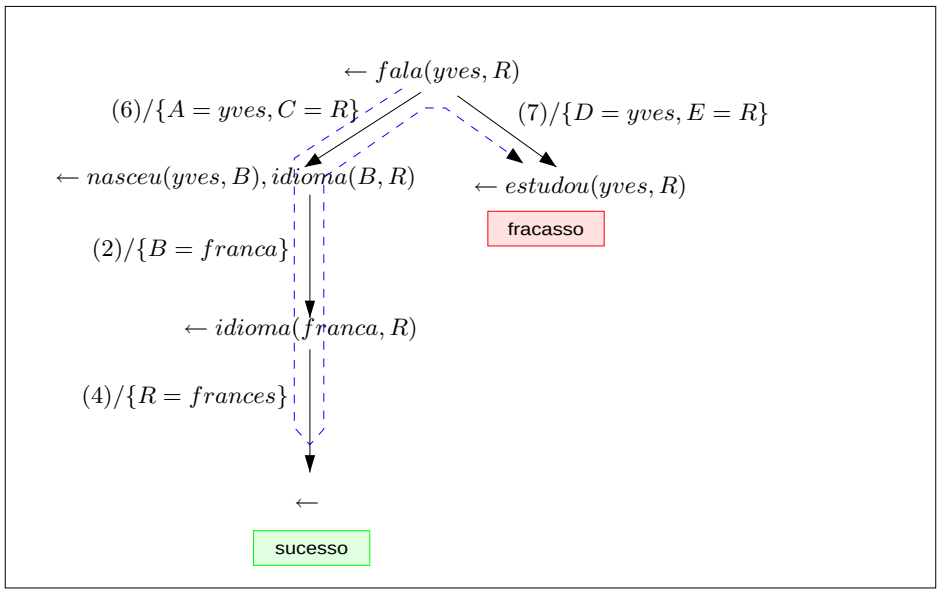

Figura 3. Árvore de refutação para a consulta $\leftarrow fala(yves, R)$

Exercício 1 *Considere o programa lógico a seguir:*

(1) $gosta(ary, eva) \leftarrow$
(2) $gosta(ivo, ana) \leftarrow$
(3) $gosta(ivo, eva) \leftarrow$
(4) $gosta(eva, ary) \leftarrow$
(5) $gosta(ana, ary) \leftarrow$
(6) $namora(A, B) \leftarrow gosta(A, B), gosta(B, A)$

Mostre como o algoritmo SLD-RESOLUÇÃO *responde às seguintes consultas:*

- *Eva namora com Ary?*
- *Ivo namora com Ana?*
- *Ary namora com quem?* □

Exercício 2 *Considere o programa lógico a seguir:*

(1) $pai(adao, cain) \leftarrow$
(2) $pai(adao, abel) \leftarrow$
(3) $pai(adao, seth) \leftarrow$
(4) $pai(seth, enos) \leftarrow$
(5) $avo(X, Z) \leftarrow pai(X, Y), pai(Y, Z)$

Mostre como o algoritmo SLD-RESOLUÇÃO *responde às seguintes consultas:*

- *Quem é avô de Enos?*
- *Seth é avô de alguém?* □

Exercício 3 *Mostre que, com base no programa a seguir, o algoritmo* SLD-RESOLUÇÃO *encontra três respostas para consulta* $\leftarrow irmao(cain, R)$.

(1) $pai(adao, cain) \leftarrow$
(2) $pai(adao, abel) \leftarrow$
(3) $pai(adao, seth) \leftarrow$
(4) $pai(seth, enos) \leftarrow$
(5) $irmao(X, Y) \leftarrow pai(Z, X), pai(Z, Y)$ □

2.3 Predicados computáveis

Predicados computáveis [5] são predicados que podem ser avaliados diretamente pelo algoritmo de refutação, sem que haja necessidade de serem definidos no programa lógico. Por exemplo, operadores aritméticos $(+, -, *, /)$ e relacionais $(=, \neq, <, \leq, >, \geq)$ são predicados computáveis. Quando um predicado computável é encontrado durante uma refutação, o algoritmo SLD-RESOLUÇÃO sinaliza fracasso apenas se a avaliação desse predicado resultar em falso.

Exemplo 3 Considere o programa lógico a seguir (que usa o predicado $\neq$):

(1) $pai(adao, cain) \leftarrow$
(2) $pai(adao, abel) \leftarrow$
(3) $pai(adao, seth) \leftarrow$
(4) $irmao(X, Y) \leftarrow pai(Z, X), pai(Z, Y), X \neq Y$

A árvore de refutação construída para responder à consulta $\leftarrow irmao(cain, R)$, com base em nesse programa, pode ser vista na Figura 4. Observe que o SLD-RESOLUÇÃO sinaliza fracasso apenas quando a avaliação do predicado $\neq$ resulta em falso; em todos os outros casos, o algoritmo sinaliza sucesso. □

Exercício 4 *Com base no programa lógico a seguir, mostre como o algoritmo* SLD-RESOLUÇÃO *responde à consulta* $\leftarrow infiel(R)$.

$gosta(ary, eva) \leftarrow$
$gosta(ivo, eva) \leftarrow$
$gosta(ary, bia) \leftarrow$
$gosta(eva, ary) \leftarrow$
$namora(A, B) \leftarrow gosta(A, B), gosta(B, A)$
$infiel(C) \leftarrow namora(C, D), gosta(C, E), D \neq E$ □

3 Negação por falha finita

Para tratar a negação, uma pequena modificação é necessária no algoritmo de refutação: quando o algoritmo encontra um literal negado, ele dispara uma refutação do literal complementar. Se essa refutação sucede, o algoritmo sinaliza `fracasso`; caso contrário, o algoritmo prossegue com a busca [4].

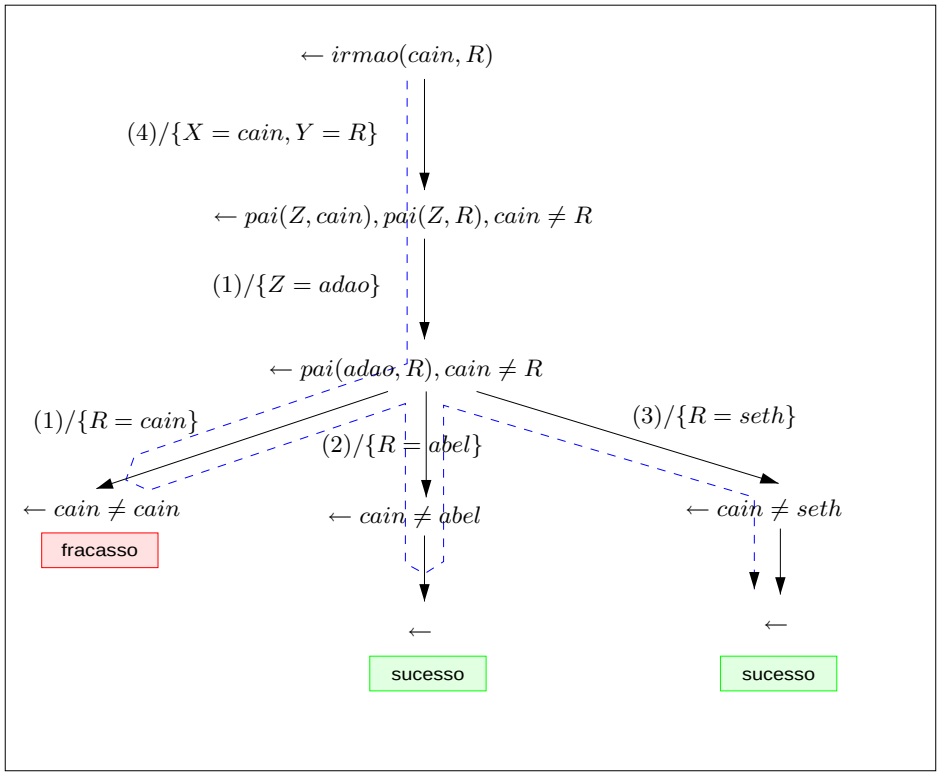

Figura 4. Árvore de refutação para a consulta $\leftarrow irmao(cain, abel)$

Exemplo 4 A árvore de refutação para responder à consulta $\leftarrow voa(R)$, com base no programa a seguir, é apresentada na Figura 5.

(1) $ave(fred) \leftarrow$
(2) $ave(bob) \leftarrow$
(3) $pinguim(fred) \leftarrow$
(4) $voa(X) \leftarrow ave(X), \neg pinguim(X)$

Nessa árvore, retângulos pontilhados representam sub-refutações. Para mostrar que "*Fred não é um pingüim*", o algoritmo tenta mostrar justamente o oposto, ou seja, que "*Fred é um pingüim*". Então, como essa sub-refutação é bem sucedida (devido à cláusula (3) do programa), a refutação inicial fracassa. Por outro lado, quando tenta mostrar que "*Bob não é um pingüim*", a sub-refutação fracassa e, portanto, a refutação inicial é bem sucedida. □

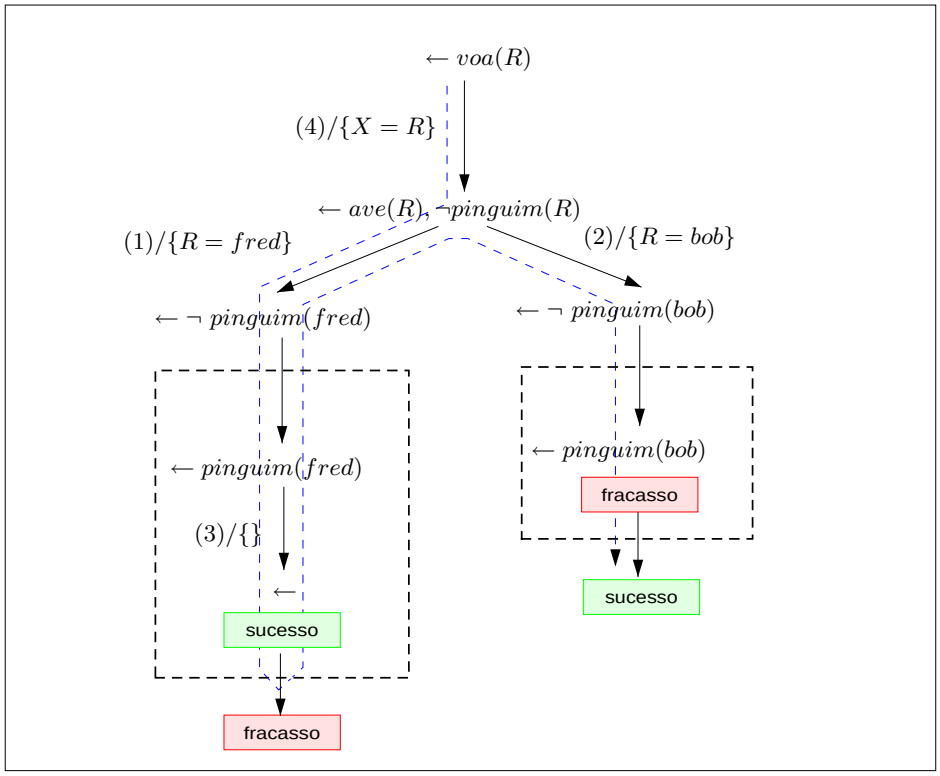

Figura 5. Árvore de refutação para a consulta $\leftarrow voa(R)$

Exercício 5 *Um fato interessante é que o predicado computável* $\neq$ *pode ser implementado através de negação por falha, conforme especificado a seguir:*

$igual(X, X) \leftarrow$
$diferente(X, Y) \leftarrow \neg\ igual(X, Y)$

Com base nessa especificação, mostre como o algoritmo SLD-RESOLUÇÃO *responde às consultas a seguir:*

(a) $\leftarrow diferente(bola, bola)$
(b) $\leftarrow diferente(bola, bala).$ □

Referências

1. AMBLE, T. *Logic Programming and Knowledge Engineering*, Addison-Wesley, 1987.
2. BRATKO, I. *Prolog - Programming for Artificial Intelligence*, Addison-Wesley, 1990.
3. GENESERETH, M. R. AND NILSSON, N. J. *Logical Fundations of Artificial Intelligence*, Morgan Kaufmann Publishers, 1988.
4. REITER, R. *On Closed Word Data Bases*, In H. Gallaire and J. Minker, Logic and Data Bases, pages 55-76, Plenum Press, NY, 1978.
5. RICH, E. AND KNIGHT, K. *Inteligência Artificial*, 2^a ed., Makron Books, 1995.