Predicate Logic

Prolog

Planner

Popler

Conniver

Lisp

Pop-2

Algol 68

Fortran

Algol

PL/1

artigo

PROLOG, linguagem de resolução de problemas pela lógica [1]

LUÍS MONIZ PEREIRA
Divisão de Informática, LNEC

RESUMO

Apresenta-se a linguagem de programação Prolog, baseada na lógica de predicados, e fornecem-se exemplos de aplicação [1].

0. INTRODUÇÃO

O Prolog (Roussel 1975, Pereira 1977) é uma linguagem de programação desenvolvida na Universidade de Marselha como instrumento prático de programação em lógica (Kowalsky 1974, Emden 1975). É especialmente útil para as aplicações que envolvem processamento simbólico e/ou pesquisa automática de alternativas ou busca sistemática de soluções possíveis (veja-se a secção Referências Bibliográficas deste número da revista).

Trata-se duma linguagem simples, mas surpreendentemente potente, baseada na Lógica de Predicado da 1.ª ordem, mas que não exige do utilizador o conhecimento dessa Lógica. Uma combinação única de características permite ao utilizador escrever programas claros, legíveis e concisos, eficientes, rapidamente e com poucos erros.

Em (Warren, Pereira, Pereira 1977) argumenta-se que em muitas aplicações o Prolog substitui com vantagem o Lisp.

1. INTRODUÇÃO À SINTAXE E NOMENCLATURA

- Um **programa** consiste num conjunto de **procedimentos**.

- Um **procedimento** é identificado pelo seu nome, a que se chama **predicado**, e pela sua **aridade** ou número de argumentos. É constituído por um certo número de cláusulas.

- Toda a **cláusula** compreende um **implicado, cabeça** ou **ponto de entrada do procedimento,** que especifica uma possível forma dos argumentos, e um **corpo,** que consiste num certo número (eventualmente zero) de **implicantes** ou **chamadas de procedimento.** Estes especificam as condições em que o implicado constitui uma afirmação verdadeira. Se o corpo for vazio referir-nos-emos a uma **cláusula unitária.**

- Exemplo de programa:

```
concatenar (nil, L, L).
concatenar (H. T, L, H. S):—
    concatenar (T, L, S).
```

Este programa é constituído por duas cláusulas, a primeira das quais unitária, que definem o procedimento cujo predicado é «concatenar», de três argumentos. Na segunda cláusula o implicado ou cabeça é separado do corpo por «:—». Toda a cláusula termina com um «.».

- Todos os objectos Prolog são **termos.**

Um **termo** é:

ou uma **variável** (começada por uma maiúscula);
ou um **átomo** (como «nil», com minúsculas apenas);
ou um **termo composto** (como «H.T»).

Um **termo composto** compreende um functor de aridade $N \geqslant 1$, e os seus N argumentos, cada um dos quais é por sua vez um termo. Por exemplo o functor de «H.T» é «.» de aridade 2, e os seus argumentos são as variáveis H e T. Está escrito em notação infixa, que é opcional. A notação standard seria «.(H, T)».

Um átomo é um functor de aridade zero.

O implicado e os implicantes de uma cláusula, bem como a própria cláusula, são termos, de forma que um predicado é apenas um functor que toma esse nome quando ocorre num certo contexto, nomeadamente quando é a cabeça de uma cláusula. Do mesmo modo, o termo (',' significa o carácter vírgula):

```
:—(p(X), ',' (q(X), r(Y))))
```

que em notação infixa se escreve

```
p(X):— q(X), r(Y)
```

será uma cláusula se figurar no conjunto das cláusulas que definem um procedimento; nesse caso é terminado por um «.».

- Aparte as convenções sintáticas, os nomes (e número) dos termos utilizados num programa são arbitrários.

- É claro que certos procedimentos já se encontram pré-definidos no sistema. Correspondem a rotinas de input/output, de aritmética, etc. Esses procedimentos têm um nome e uma aridade standard.

2. SEMÂNTICA

- O Prolog difere doutras linguagens de programação no facto de haver dois modos distintos de entender a sua semântica.

- A semântica **procedimental** descreve, segundo é usual, a sequência dos estados por que passa a execução de um programa.

- Além disso, um programa Prolog pode ser entendido como um conjunto de afirmações acerca de um problema. A semântica **declarativa,** que o Prolog herda da Lógica, proporciona uma base formal para essa leitura.

2.1 — Semântica declarativa

- Informalmente, interpreta-se cada termo como uma abreviaura duma expressão em língua natural, por aplicação uniforme duma tradução para cada functor. Cada variável é interpretada como um objecto arbitrário.

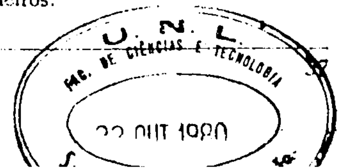
Exemplo:

nil=«a lista vazia»

H.T=«a lista cujo primeiro elemento é H e restantes elementos constituem a sublista T»

---

[1] Esta primeira parte foi comunicada ao Congresso 77, realizado em Lisboa, Novembro de 1977, e organizado pela Ordem dos Engenheiros.



concatenar (nil, L, L) = «L é o resultado de concatenar a lista vazia com L»
concatenar (L1, L2, L3) = L3 é o resultado de concatenar L2 com L1»

- Uma cláusula da forma

$$P{:}-Q,\ R,\ S.$$

é interpretada como

«P se Q e R e S».

De forma que a cláusula

concatenar (H. T, L, H. S): —
concatenar (T, L, S).

se interpreta: «a lista cujo primeiro elemento é H e restantes elementos constituem a sublista S é o resultado de concatenar L com a lista cujo primeiro elemento é H e restantes elementos constituem a sublista T se S for o resultado de concatenar T com L».

- Por outro lado, a semântica declarativa define (recursivamente) o conjunto dos termos que se podem afirmar verdadeiros de acordo com um programa:

Um termo é **verdadeiro** se for identificável com o implicado de alguma **instância** de uma cláusula, e se cada um dos implicantes (se os houver) dessa instância for verdadeiro.

Uma **instância** de uma cláusula (ou em geral de um termo) é a cláusula (termo) que se obtém substituindo, para cada uma de zero ou mais variáveis, todas as ocorrências dessa variável por um novo termo.

Exemplo:

p(g(W, k), a, V): — q(g(W, k), b, f(g(W, k), c)).

é uma instância de

p(X, a, Y): — q(X, b, f(X, Z)).

onde X foi substituído por g(W, k), Y por V, e Z por c.

- É o aspecto declarativo da semântica do Prolog que proporciona uma programação clara, rápida e precisa, por permitir a subdivisão de um programa em pequenas unidades (cláusulas) individualmente significativas, cujos termos componentes são também igualmente significativos, etc.

- Por outro lado, a sua semântica declarativa (formal) possui o rigor que lhe advém de ser interpretada na Lógica de Predicados da 1.ª ordem.

2.2. — **Semântica declarativa formal** (²)

- Um programa Prolog é a **conjunção lógica** das suas cláusulas.

- Cada cláusula exprime uma **proposição lógica** em que o implicado (ou cabeça da cláusula) **é logicamente implicado** pela **conjunção lógica** dos implicantes do seu corpo. Todas as variáveis da cláusula estão implicitamente quantificadas universalmente.

Exemplo:

$$\forall\ X, Y\ \ (p(X, a) \rightarrow q(Y))$$

exprime-se

$$q(Y){:}-p(X, a).$$

- Só as proposições do tipo Horn são redutíveis à forma canónica de uma implicação quantificada universalmente em todas as suas variáveis. Isto é, as do tipo

$$\forall\ X_1, \ldots, X_n\ (A_1 \wedge \ldots \wedge A_j \rightarrow B)$$

São proposições do tipo Horn as proposições sem variáveis livres que quando reduzidas à sua Prenex Normal (o que é sempre possível) na sua forma dijuntiva, apresentam quando muito um átomo (no sentido lógico) não negado.

Em particular, uma proposição com Prenex Normal Form

$$\forall\ X_1,\ldots,X_n\ (\sim A_1 \vee \ldots \vee \sim A_j \vee B)$$

É equivalente a

$$\forall\ X_1,\ldots,X_n\ (A_1 \wedge \ldots \wedge A_j \rightarrow B)$$

e portanto exprime-se em Prolog

$$B{:}-A_1, \ldots, A_j.$$

- Demonstra-se que com proposições Horn se pode exprimir qualquer função computável, e na prática verifica-se que é rara a necessidade de exprimir um problema com cláusulas não-Horn, embora haja um interpretador escrito em Prolog capaz de lidar com essas cláusulas.

- Os termos que figuram no lugar de argumentos geram um universo de Herbrand (cf: por ex. (Chang, Lee 1973)).

- Os predicados geram uma base de Herbrand.

- A operação de identificar um termo com o implicado de uma instância de uma cláusula corresponde ao processo da **unificação.**

A **unificação** faz parte do **princípio da resolução,** que é a regra de inferência única segundo a formalização da Lógica de Predicados de 1.ª ordem feita por Robinson (Robinson 1965).

- Basicamente, o **princípio da resolução** reúne numa única regra de inferência as duas regras «clássicas», «modus ponens» e «generalização».

Permite por exemplo a partir de

$$A(X) \leftarrow B(a, X),\ C(X)$$

e

$$B(Y, f(Y, Z)) \leftarrow D(Y)$$

concluir

$$A(f(a, Z)) \leftarrow D(a),\ C(f(a, Z))$$

- Note-se a vantagem, em termos de execução por computador, de haver apenas uma regra de inferência uniformemente aplicável.

- O processo de unificação fornece a **instância comum mais geral** entre dois termos, que se existe é única. A obtenção da instância comum mais geral garante que a regra de inferência tira a conclusão mais geral possível das suas duas premissas.

2.3 — **Semântica Procedimental**

- A semântica procedimental descreve o modo como um implicante **é executado** ou **resolvido.** O objectivo da execução é o de produzir instâncias verdadeiras do implicante.

- É implicante notar que a ordenação das cláusulas de um procedimento dentro de um programa, bem como a ordenação dos implicantes numa cláusula, sendo irrelevantes para a semântica declarativa pois a elas não faz referência, constituem **informação de controlo** crucial para a semântica procedimental.

(²) Pode omitir-se sem perda de continuidade.

- Para **executar** ou **resolver** um implicante P, o sistema de execução procura a primeira cláusula cujo implicado se **adequa** ou **unifica** com P. O processo de unificação fornece, como já mencionámos, a instância comum mais geral entre os dois termos, implicante e implicado, que se existir é única. Se ela existir, a instância de cláusula obtida é **activada** através da execução sucessiva, da esquerda para a direita, de cada um dos implicantes do seu corpo (se os houver).

- Se o sistema não encontra uma cláusula adequada a um implicante, esse implicante **falha** e o sistema efectua um **retrocesso.** Isto é, rejeita a cláusula mais recentemente activada, desfazendo todas as substituições que resultaram da unificação com o implicado da cláusula. Seguidamente reconsidera o implicante que originalmente a activou, e procura uma cláusula subsequente, com o mesmo predicado, que também se adeque a esse implicante.

- A execução é bem sucedida quando já não existirem mais implicantes à espera de serem executados. (O sistema encontrou uma instância do implicante original P que é verdadeira). A execução é mal sucedida quando o retrocesso regressa até ao implicante original P e já não existem mais escolhas possíveis de cláusulas que se adequem a P. É claro que a execução pode não terminar.

- Além da sequenciação das cláusulas e dos implicantes, o Prolog proporciona um outro mecanismo muito importante para especificar informação de controlo. Trata-se do símbolo de «corte», que se escreve «!». Insere-se num programa do mesmo modo que um implicante, mas não deve no entanto considerar-se parte da lógica do programa, devendo ignorar-se do ponto de vista da semântica declarativa.

- O efeito do símbolo do «corte» é como segue. Quando encontrado pela primeira vez como «implicante», um «corte» resolve imediatamente, passando a execução ao implicante seguinte. Caso o retrocesso volte mais tarde a esse «corte», o efeito deste será o de falhar o implicante que provocou a activação da cláusula que contém o «corte». Por outras palavras, um «corte» obriga o sistema a todas as escolhas feitas desde que o implicante activador começou a ser executado. Isto é, não se consideram outras alternativas nem para esse implicante nem para os implicantes da cláusula que estão antes do «corte».

- Por meio do «corte» pode pois impor-se que certos implicantes, uma vez parcialmente executados, ou continuam a executar-se por essa via ou falham.

- Exemplo do efeito de «!».

P: — A, B, C.

B: — D, E, !, F.

B: — G, H.

As setas traduzem o fluxo de controlo. Este ao ser transmitido a P é sucessivamente passado a A e a B. Suponhamos que A resolveu com uma cláusula A1 que não mostramos. B vai de seguida activar a primera cláusula para B. Uma vez resolvidos D e E, com cláusulas D1 e E1 suponhamos, «!» resolve e o controlo passa a F. Imaginemos que F falha. O retrocesso leva a «!», que provoca um retorno do controlo para A, que irá em seguida tentar resolver com a cláusula A2. Isto é, não são tentadas outras resoluções para D, E ou B, uma vez que D e E estão na cláusula com o «corte» e antes deste, e uma vez que B foi quem activou essa cláusula. Assim o controlo passa para o ponto imediatamente anterior a B, que neste caso é A. Se A não existisse o controlo era passado ao implicante que activou a cláusula P.

3. EXEMPLOS DE PROGRAMAS

3.1 — Distribuição de "*" por "+"

Este programa distribui os factores pelas parcelas numa expressão formada pelos functores binários "*" e "+".

```
multiplicar (X+Y, Z, X1+Y1): —

        multiplicar (X, Z, X1),
        multiplicar (Y, Z, Y1).

multiplicar (Z, X+Y, X1+Y1): —

        multiplicar (Z, X, X1),
        multiplicar (Z, Y, Y1).

multiplicar (X, Y, X * Y).
```

A primeira cláusula efectua a distributividade à direita, e a segunda à esquerda.

A última cláusula, que só é utilizada se as duas primeiras não se aplicarem, limita-se a colocar "*" entre os dois factores, indecomponíveis em parcelas pelas cláusulas anteriores.

3.2 — Programa para numerar por ordem alfabética os elementos de uma lista

Dada a lista p.r.o.l.o.g.nil, pretende-se gerar 4.5.3.2.1.nil. Comecemos por conceber um programa para **verificar** se uma lista resultado constitui uma solução correcta — uma táctica característica no Prolog.

O programa assim obtido **gera** a lista resultado no caso de ela não ser conhecida. Chamemos «alfabetizar» à operação de numerar por ordem alfabética os elementos de uma lista dada L para obter a lista R com essa numeração. A «alfabetização» pode concluir-se em três etapas:

1.º — emparelhar os elementos da lista dada L com os elementos da lista resultado R para obter uma lista A de pares.

2.º — compor os pares assim obtidos por ordem alfabética numa árvore binária T.

3.º — numerar os nós da árvore desde 1 até N.

```
alfabetizar (L, R): —

    emparelhar (L, R, A),
    compor (A, T),
    numerar (T, 1, N).

emparelhar
    (X . L, Y . R, par (X, Y) . A): —

        emparelhar (L, R, A).

emparelhar (nil, nil, nil).
```

Vamos representar uma árvore binária por «árvore (T1, X, T2)» onde X é o nó raiz da árvore e T1 e T2 são as subárvores esquerda e direita, respectivamente, que pendem de X.

Para criar a árvore duma lista Y.L de nós, vamos partir L em L1 e L2 relativamente a Y. L1 contém todos os elementos de L menores que Y, e L2 todos os maiores. Seguidamente vamos com L1 e L2 compor as árvores T1 e T2. Quando L é a lista vazia obtemos a árvore «vazia».

```
compor (Y. L, árvore T1 Y, T2)): —

        partir (L, Y, L1,L2),
        compor (L1, T1),
        compor (L2, T2).

compor (nil, vazia).
```

Para «partir» L em L1 e L2 em ordem a Y temos quatro casos a considerar. Se um elemento de L coincide com Y continua-se a partir L ignorando esse elemento. Se é anterior a Y coloca-se em L1 e continua-se a partir. Se Y é anterior a ele coloca-se em L2 e continua-se a partir. Quando L é vazia, L1 e L2 são também vazias.

Um par é anterior a outro se a letra do primeiro for alfabeticamente inferior, «<», à letra do segundo. Supomos «<» um predicado binário pré-existente na linguagem.

```
partir (Y.L, Y, LE, L2): —
    partir (L, Y, L1, L2).

partir (X.L, Y, L1, X.L2): —
    anterior (X, Y),
    partir (L, Y, L1, L2).

partir (X.L, Y, X.L1, L2): —
    anterior (Y, X),
    partir (L, Y, L1, L2).

partir (nil, Y, nil, nil).

anterior (par (X1,Y1), par (X2, Y2))
    : — X1 < X2.
```

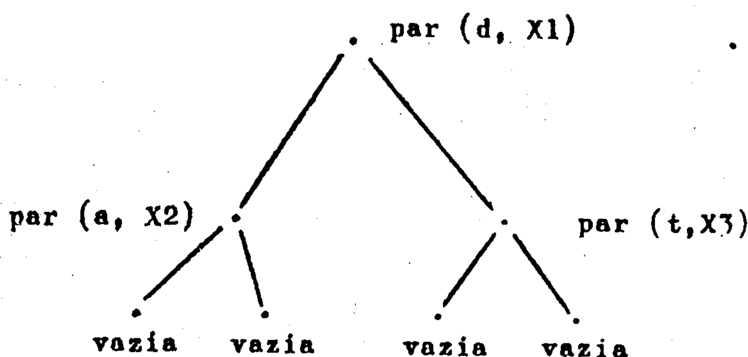
Para o problema

«alfabetizar (d.a.t.a. nil, R)»

temos depois de emparelhar:

```
R=X1. X2. X3. X4. nil
A=par (d, X1). par (a, X2).
    par (T, X3). par (a, X4). nil
```

Depois do compor:



onde a variável X4 foi entretanto feita igual à variável X2, por unificação, pela primeira cláusula «partir».

Para numerar uma árvore binária a partir de N0 até N (exclusive) começamos por numerar a subárvore esquerda T1 desde N0 até N1, numeramos a raiz com N1, e finalizamos numerando a subárvore direita desde N2 até N. Se a árvore é vazia a numeração não sofre alteração:

```
numerar (árvore (T1, par (X, N1), T2),
   N0, N): —

      numerar (T1, N0, N1),
      N2 is N1+1,
      numerar (T2, N2, N).

numerar (vazia, N, N).
```

«is» é um predicado binário, pré-definido na linguagem, que unifica o primeiro argumento com o resultado da expressão aritmética fornecida no segundo argumento.

3.3 — **Programa para derivação** simbólica

O primeiro argumento recebe a expressão a derivar, a qual é formada por identificadores (ou sequências de letras) e números apenas, combinados pelos functores «+», «—» unário, «—» binário, «*», «/», «↑», «log», «exp».

O segundo argumento recebe o identificador que representa a variável em relação à qual se pretende derivar (esse identificador do ponto de vista Prolog é um átomo e não uma variável). O terceiro argumento fornece a expressão derivada não simplificada. Os predicados «constante (X)» e «X=/=Y» pressupõem-se já definidos. O primeiro resolve se X for um átomo ou um número; o segundo resolve se X for diferente de Y.

```
derivada (X, X, 1).
derivada (K, X, 0): — constante (K), K=/=X.
derivada (U + V, X, DU + DV); — derivada (U, X, DU), derivada (V, X, DV).
derivada (U - V, X, DU - DV); — derivada (U, X, DU), derivada (V, X, DV).
derivada (—U, X,-DU): — derivada (U, X, DU).
derivada (K * U, X, K * DU): — constante (K), derivada ((U, X, DU).
derivada (U * V, X, DV * U+DU * V): — derivada (U, X, DU),
                                      derivada (V, X, DV).
derivada (U/V, X, (DU * V—DV * U)/V ↑ 2): — derivada (U, X, DU).
                                            derivada (V, X, DV).
derivada (U ↑ V, X, V * DU * U ↑ (V— 1 )) — constante (V),
                                            derivada (U, X, DU),
derivada (U ↑ V, X, DV * log (U) * U ↑ V+V * DU * U ↑ (V-1) : --
                                            derivada (U, X, DU).
                                            derivada (U, X, DV).
derivada (log U, X, DU/U): — derivada (U, X, DU).
derivada (exp U, X, DU * exp U): — derivada (U, X, DU).
derivada (sen U, X, DU * cos U): — derivada (U, X, DU).
derivada (cos U, X, —DU * sen U): — derivada (U, X, DU).
derivada (tg U, X, D) : — derivada (sen U/cos U, X, D).
```

3.4 — Bases de dados

O seguinte exemplo mostra a identidade entre programa e dados no Prolog, e mostra a potencialidade da linguagem como meio natural de interrogação de bases de dados. Uma «base de dados» de cláusulas unitárias fornece informações (em milhões) e áreas (em milhares de milhas quadradas) de vários países. O procedimento «densidade» fornece «dados virtuais» sobre a densidade de população:

```
pop (china, 825).          área (china, 3380).
pop (índia, 586).          área (índia, 1139).
pop (urss, 252).           área (urss, 8708).
pop (usa, 212).            área (usa, 8609).
.
.
.
densidade (C, D): — pop (C, P), área (C, A).
                    D is (P * 1000)/A.
```

A cláusula seguinte representa uma interrogação à base de dados para encontrar os países com densidade de população semelhante, isto é, que difiram menos de 5 %:

```
semelhantes (C1, D1, C2, D2): — densidade (C1, D1),
                                densidade (C2, D2),
                                D1 > D2,
                                20 * D1 < 21 * D2.
```

NOTA: «<», «*», «/» e «is» são predicados calculáveis pelo sistema. São respectivamente «menor», «vezes», «divisão inteira» e «atribuição de valor».

4 — REFERÊNCIAS

CHANG, C.; LEE, R. C. — 1973 — «Symbolic Logic and Mechanical Theorem Proving». Academic Press, 1973.

VAN EMDEN, M. H. — 1975 — «Programming with resolution Logic». Report CS-75-70, Dept. of Computer Science, University of Waterloo, Canada, Nov. 1975.

KOWALSKY, R. A. — 1974 — «Logic for problem solving DCL». Memo 75, Dept. of Artificial Intelligence, University of Edinburgh. Mar. 74.

PEREIRA, L. M. — 1978 — «User's guide to DECsystem-10 Prolog.». Laboratório Nacional de Engenharia Civil, Lisboa.

ROBINSON, J. A. — 1965 — «A Machine oriented logic based on the resolution principle». J. Association for Computing Machinery, vol. 12, pp. 23-45, 1965.

ROUSSEL, P. — 1975 — «Prolog: Manuel de référence et d'utilisation». Groupe d'Intelligence Artificielle, Marseille — Luminy. Set. 75.

WARREN, D. H. D.; PEREIRA, L. M.; PEREIRA, F. — 1977 — «Prolog — the language and its implementation compared with Lisp». Laboratório Nacional de Engenharia Civil, Lisboa.

PROLOG, LINGUAGEM DE RESOLUÇÃO DE PROBLEMAS PELA LÓGICA — PARTE 2 [1]

Luís Moniz Pereira
Universidade Nova de Lisboa

1. MODELO LÓGICO DA RESOLUÇÃO DE PROBLEMAS

O modelo de resolução de problemas que vamos apresentar de seguida usa a representação em árvore E/OU.

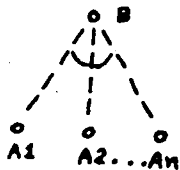
Dado um problema inicial, dado o conjunto de operadores que reduzem problemas a sub-problemas, e dado um conjunto de sub-problemas já resolvidos ou factos, pretendemos resolver o problema inicial pela aplicação sucessiva dos operadores a esse problema e seus sub-problemas, até que todos os sub-problemas por resolver coincidam com ou seja casos particulares de factos já conhecidos.

Teremos sucesso se conseguirmos reduzir o problema inicial a um conjunto dos sub-problemas resolvidos.

1.1 Caso de Problemas Independentes

Na representação em árvore E/OU, a cada nó corresponde um problema:

(1) a raiz é rotulada com o problema inicial
(2) se um nó é rotulado por um problema B, e existe um operador que reduz B aos sub-problemas A1,...,An então o nó B é ligado por um arco dirigido para cada um dos n nos sucessores, rotulados individualmente pelos sub-problemas A1,...,An



Os arcos dos sucessores de um mesmo operador são ligados entre si para que na representação em árvore se distingam quais os sucessores determinados por diferentes operadores.

Os problemas inicialmente já resolvidos correspondem a nós com um único sucesso, rotulado pela marca de sucesso "[]".

Uma resolução do problema inicial é uma sub-àrvore finita duma àrvore E/OU — que

(1) contém a raiz
(2) os sucessores de todo o nó não não rotulado com [] resultam da aplicação de um único operador

Utilizaremos a notação

←A1,...,An

para indicar que A1,...,An são problemas a resolver; a notação

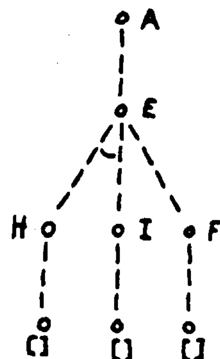
B ← A1,...,An

para indicar que existe um operador que reduz o problema B aos sub-problemas A1,...,An, e a notação

C ←

para indicar que C é um problema inicialmente resolvido ou facto.

A figura seguinte ilustra o espaço de resoluções para o problema A, representado por uma àrvore E/OU, e resulta de se considerarem os operadores e os factos que abaixo se apresentam:



Problema inicial a resolver:

← A

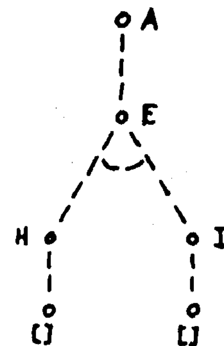
Operadores:

A ← E
E ← H, I
E ← F

Problemas iniciais resolvidos ou factos:

H ←
I ←
F ←

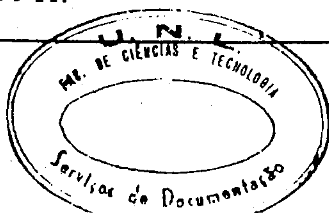
Existem duas resoluções para o problema A:



que corespondem aos dois modos diferentes de resolver o problema E.

Vejamos como decorre a primeira resolução de A. Partindo de

(1) A primeira parte deste artigo foi publicada na "Informática" VOL. 2, n.º 4, Agosto/Setembro 1978, pág. 37-41. As referências bibliográficas encontram-se no mesmo número, págs. 9-11.


U. N. L.
FAC. DE CIÊNCIAS E TECNOLOGIA
Serviços de Documentação

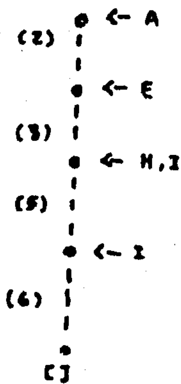
← A

e aplicando o operador A ← E, substituimos o problema a resolver ← A pelo problema ← E.

Em seguida, escolhemos o operador E ← H,I para aplicarmos a ← E, e obtermos ← H,I como problemas a resolver. No entanto, tanto H como I são problemas já resolvidos (ou factos). Assim, num primeiro passo, a partir de ← H,I obtemos ← I e por fim, apenas ←, isto é nenhum problema por resolver. ← A foi portanto completamente resolvido.

Esta mesma resolução pode ser indicada do modo seguinte

(1) ← A
(2) A ← E
(3) E ← H,I
(4) E ← F
(5) H ←
(6) I ←
(7) F ←



onde na passagem de ← A para ← E utilizámos (2), na passagem de ← E para ← H,I utilizamos (3), etc..

1.2 Caso de Problemas Dependentes

Na secção anterior, a resolução de cada problema era independente das dos problemas anteriores. Isto é, nenhum problema recebia dos anteriores qualquer informação, e a sua resolução era independente de quais tivessem sido os problemas resolvidos anteriores e de quais tivessem sido as suas resoluções.

Vamos seguidamente mostrar a necessidade de passagem de informação duns problemas para os outros, e como podemos dar conta da interdependência dos problemas.

Suponhamos então que pretendiamos resolver o problema

(8) ← irmão (Maria, Y)

que consiste em encontrar o nome de um irmão Y de Maria, dado que conhecemos os seguintes factos

(9) mãe (Pedro, Zulmira) ←
(10) mãe (João, Zulmira) ←
(11) mãe (Maria, Zulmira) ←
(12) pai (Maria, Ernesto) ←
(13) pai (Pedro, Jacinto) ←
(14) pai (João), Ernesto) ←

e dado que o nosso conhecimento das relações de família nos diz que

(15) irmão(X,Y) ← pais(X,P,M), pais(Y,P,M), diferentes (X,Y)

isto é, que X e Y são irmãos se X e Y forem diferentes, e se tiverem os mesmos pais, P e M.

Por outro lado sabemos que

(16) pais(X,P,M) ← mãe(X,M), pai(X,P)

significando

P e M são pais de X se M for mãe de X eP pai de X.

Para resolvermos o problema proposto comecemos então por (8) e apliquemos-lhe o operador (15), obtendo (17)

(17) ← pais (Maria, P,M), pais(Y,P,M), diferentes(Maria,Y)

onde da comparação de irmão(Maria, Y) de (8), com irmão(X,Y) de (15), resultou para a variável X de (15) o valor Maria, o qual se transmitiu às outras ocorrências da variável X em (15); resulta por isso que o valor a encontrar para a variável Y em (15) terá que ser tal que

diferente(Maria,Y)

se venha a verificar. Do mesmo modo, os valores para P e M a encontrar terão que ser tais que se verifique

pais(Maria, P,M)

Prosseguindo na resolução do problema inicial, o sub-problema seguinte a resolver é o de encontrar um pai P e uma mãe M para Maria; isto é temo a descobrir para as variáveis P e M va lores que satisfaçam a relação esti pulada por

pais(Maria,P,M)

Para isso vamos aplicar a (17) o operador (16), obtendo como resultado(18)

(18) ← mãe(Maria,M), pai(Maria, P), pais(Y,P,M), diferentes(Maria,Y)

que diz afinal que uma vez encontrados uma mãe M e um pai P para Maria, resta-nos encontrar um indivíduo Y com os mesmos pais P e M, o qual não seja a própria Maria. Sobre esse indivíduo, ainda por determinar, poderemos então dizer que é irmão de Maria.

A resolução prossegue então usando em (18) o facto (11) para obter (19)

(19) ← pai(Maria,P), pais(Y,P,Zulmira), diferentes(Maria,Y)

e entrando em (19) com o facto (12) para obter (20)

(20) ← pais(Y, Ernesto,Zulmira), diferentes(Maria,Y)

O nosso próximo sub-problema a resolver é o de encontrar um indivíduo Y tal que se verifique

pais(Y, Ernesto,Zulmira)

A resolução deste sub-problema vai fazer uso dos valores já obtidos para P e M durante a resolução dos sub-problemas anteriores. Dito doutra maneira, os valores possíveis a obter agora para Y estão já condicionados pelos valores previamente obtidos para P e M pelos sub-problemas anteriores; isto é, as variáveis P e M presentes nos operadores (15) e (16) exprimem e estabelecem, em cada operador, a interdependência entre os seus sub-problemas e, além disso, durante a resolução, transmitem de uns sub-problemas para os outros, dentro do mesmo operador e entre operadores, os valores específicos de interdependência apropriados ao problema inicial específico e factos conhecidos específicos deste problema.

Prosseguindo, voltemos agora a usar o operador (16) aplicando-o a (20) para obter (21)

(21) ← mãe(Y,Zulmira),
pai(Y, Ernesto),
diferentes(Maria,Y)

Note-se que da comparação de pais (Y,Ernesto,Zulmira) de (20) com pais (X,P,M) de (16), resultou a identificação da variável X de (16) com a variável Y de (20), e a atribuição dos valores Ernestto e Zulmira, respectivamente, às variáveis P e M de (16).

Chamaremos unificação a este confronto e subsequente adequação, quando tal é possível, entre os argumentos de um sub-problema e os argumentos de um operador que lhe é aplicado ou de um facto que o resolve. Chamaremos substituição adequadora ao conjunto das atribuições executadas pela unificação entre esses argumentos.

Retomando a resolução do problema, verificamos agora que podemos utilizar em (21) quer o facto (9) quer o facto (10) quer o (11). Da consideração de cada um desses três casos resultam respectivamente as seguintes três possibilidades de sub-problemas a resolver para continuarmos a nossa procura de uma solução para o problema inicial.

(22) ← pai(Pedro, Ernesto),
diferentes(Maria,Pedro)

(23) ← pai(João, Ernesto),
diferentes(Maria,João)

(24) ← pai(Maria,Ernesto),
diferentes(Maria,Maria)

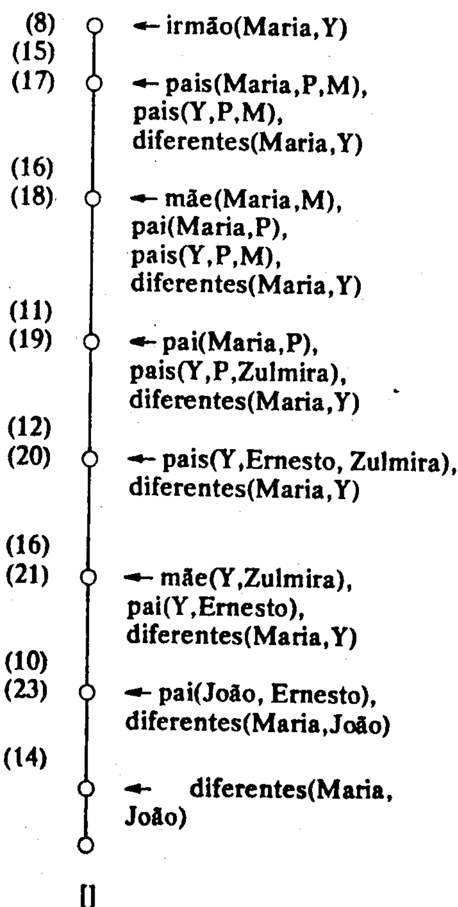
Dessas três, apenas uma conduz a uma solução. Vejamos porquê: embora a (24) se aplique (12), encontramo-nos de seguida na impossibilidade de resolver diferentes(Maria, Maria); por outro lado, não podemos continuar resolvendo (22) pois o facto

pai(Pedro,Ernesto)

de que precisaríamos, não se verifica; assim, resta-nos a hipótese de resolvermos (23).

Com a ajuda de (14) obtemos ← diferentes(Maria, João), que tomamos como facto, concluíndo deste modo a resolução do problema inicial com a resposta de que Maria tem de facto um irmão, que é João.

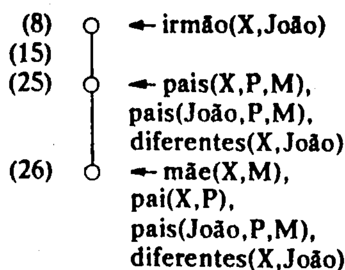
A resolução encontrada foi



Se o problema inicial for

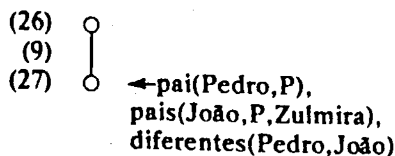
(8a) ←irmão(X,João)

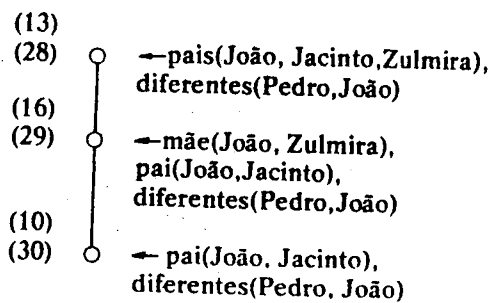
isto é, se fornecermos o resultado do exemplo anterior, João, como dado, vamos obter como resultado Maria, o dado do problema anterior.



Chegados a este ponto verificamos que a (26) podemos aplicar quer (9) quer (10) quer (11).

Se aplicarmos (9), a resolução prossegue com

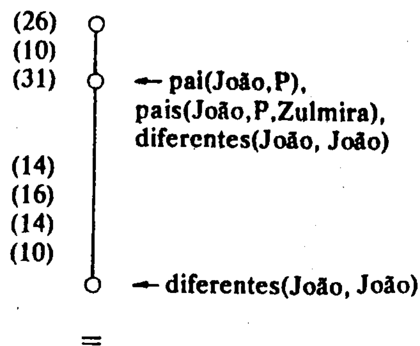




onde o símbolo "=" indica insucesso.

O insucesso resulta de não ser possível aplicar a (30) nenhum operador ou facto. Resta-nos pois voltar atrás a (26), ao último ponto onde houve uma opção de continuação da resolução, e tentarmos as restantes alternativas possíveis em aberto nesse ponto.

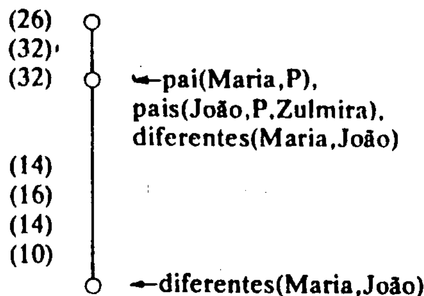
Ora se a (26) aplicarmos (10), a resolução prossegue com



onde em (10) voltamos a deparar com um insucesso pois não é possível resolver

diferentes(João,João)

Considerando por fim a aplicação a (26) de (11), obtemos



que termina com sucesso, e indicando que Maria e João são irmãos.

De igual maneira, se o problema inicial não especificasse nenhum dado particular, isto é se fosse

← irmão(X,Y)

poderíamos obter como resultado todos os pares (X,Y) que verificassem a relação "irmão". Tais pares, no contexto por exemplo dos dados de parentesco correspondentes a uma certa família, e tomados no seu conjunto, pode dizer-se que definem a relação "irmão".

Tal definição de uma relação, por enumeração dos elementos que a verificam, constitui uma "definição em extensão" dessa relação, por contraposição à especificação dessa relação através dos operadores que a calculam, a qual constitui uma "definição em intensão" dessa relação.

Ora o modelo lógico da resolução de problemas que temos vindo a considerar, permite precisamente calcular relações. Isto é, permite especificar uma relação em extensão, e permite também especificar uma relação em intensão, através dos operadores que a definem em termos das suas propriedades e/ou que a definem por intermédio de outras relações, porventuras especificadas em extensão.

Tal facto assenta na justificação ou interpretação que se pode efectuar, segundo a Lógica de Predicados, do modelo de resolução de problemas apresentado, uma vez que um predicado não é mais que a indicação de uma relação a verificar-se entre os seus argumentos.

Por isso se torna possível nos exemplos exibidos resolver problemas quer da forma " encontre-me um irmão para fulano", como ainda "encontre-me fulanos e sicranos que sejam irmãos", e ainda "verifique se tal fulano e tal sicrano são irmãos".

Este último caso corresponderia a ir resolver por exemplo

← irmão(Maria,Pedro)

ou

← irmão(Maria,João)

Voltando ao último problema resolvido, iremos chamar ao mecanismo nele ilustrado que consiste em voltar atrás no processo de resolução para examinar alternativas pendentes, "mecanismo de retrocesso" ou simplesmente "retrocesso" (backtracking).

O modo de efectuar o retrocesso define, conjuntamente com o modo de sequênciar a selecção de operadores e factos para o avanço da resolução, a chamada "estratégia de procura" ou simplesmente "estratégia".

2. LINGUAGENS DEDUTIVAS

As linguagens dedutivas são linguagens de programação que, dado um conjunto de hipóteses iniciais e um conjunto de regras lógicas de manipulação, permitem explorar as conclusões lógicas possíveis resultantes da aplicação dessas regras e suas combinações às hipóteses e conclusões lógicas intermédias.

Basicamente, as hipóteses iniciais podem interpretar-se como FACTOS conhecidos, e as regras lógicas como MÉTODOS que indicam como concluir NOVOS FACTOS a partir de factos já conhecidos.

Uma outra possibilidade consiste em interpretar uma hipótese inicial como uma RESPOSTA CONHECIDA, e uma regra lógica como um MODO de reformulação de uma pergunta em termos de outras. Assim, o averiguar de uma conclusão lógica a partir das hipóteses interpretar-se-á como a tentativa de encontrar para essa conclusão lógica encarada como PERGUNTA INICIAL, uma resposta, encontrando para isso uma sequência de modos de reformulação dessa pergunta em termos de outras, (e de reformulação destas, etc.), até que não seja mais possível reformular as perguntas que sobrem em termos de outras. Nessa altura, se conhecermos uma RESPOSTA para cada uma dessas perguntas, teremos encontrado resposta completa para a pergunta inicial, isto é, teremos respondido conclusivamente à pergunta inicial.

Outra interpretação ainda pode obter-se encarando cada hipótese inicial como um PROBLEMA JÁ RESOLVIDO, cada regra lógica como um PROCESSO DE RESOLUÇÃO de um problema por redução aos seus sub-problemas. Assim, o averiguar de uma conclusão lógica a partir das hipóteses interpretar-se-á como a tentativa de encontrar para essa conclusão lógica, encarada como PROBLEMA INICIAL, uma solução encontrando uma sequência de processos de resolução para esse problema e sub-problemas resultantes etc., até que todos os problemas por resolver sejam PROBLEMAS JÁ RESOLVIDOS. Nessa altura, teremos encontrado uma solução para o problema inicial, isto é, tê-lo-emos resolvido assim como os sub-problemas intermédios gerados.

A concluir, interessa notar os seguintes pontos:

(1) as linguagens dedutivas visam permitir exprimir nelas as interpretações referidas e similares.

(2) uma vez feita a interpretação pretendida, a própria linguagem, através do sistema que por sua vez a interpreta' se encarrega de executar a procura das DEDUÇÕES que, a serem possíveis, conduzam às conclusões pretendidas.

(3) o resultado dessa actividade, no caso de haver sucesso, será não apenas o estabelecimento da conclusão como facto, nos casos de resposta a uma pergunta ou de resolução de um problema, mas também e respectivamente, na explicitação da sequência das conclusões relativas às respostas intermédias que foi necessário responder ao longo da explicitação das perguntas intermédias formuladas, e na explicitação das soluções dos problemas intermédios suscitados. Numa frase, atinge-se o objectivo inicial e explicita-se também o modo de o atingir.

De entre as linguagens dedutivas destacam-se as que se baseiam na LÓGICA PREDICATIVA. Nessas, as hipóteses e as regras lógicas surgem como AXIOMAS, expressos numa notação próxima da linguagem da lógica de predicados de primeira ordem, e na qual a análise de uma conclusão lógica consiste em averiguar se ela é um TEOREMA duma TEORIA: aquela que é definida pelos AXIOMAS dados e pelas REGRAS DE INFERÊNCIA da Lógica Predicativa. O papel de tais linguagens dedutivas é o procurar DEMONSTRAÇÕES para os TEOREMAS que se lhes propõe.

São estas linguagens dedutivas DEMONSTRATIVAS (e em particular baseadas no PRINCÍPIO da RESOLUÇÃO) que temos abordado aqui, em particular a linguagem de progrmação PROLOG.

O uso que determinada aplicação fará duma linguagem dedutiva demonstrativa como o PROLOG está essencialmente dependente da possi-

bilidade de interpretar ou re-interpretar os problemas e o domínio do conhecimento que essa aplicação pretende abordar, reformulando-os adentro duma das perspectivas mencionadas acima ou outras semelhantes, como é o caso das regras de re-escrita.

É nessa correspondência entre a linguagem em que a aplicação se expressa e o formalismo posto à disposição pela linguagem dedutiva que reside a condição básica de passagem à resolução automática de uma dada aplicação.

Tal correspondência implica òbviamente um diálogo entre as duas partes.

3. ALGUMAS VANTAGENS DO PROLOG

Uma maior vantagem do PROLOG relativamente as linguagens tipo PLANNER consiste na sua extrema simplicidade, o que lhe permite ser aprendida mais depressa. Um programa pode ser escrito em PROLOG mais facilmente e com menos erros, e pode ser modificado e desenvolvido com menor dificuldade.

A simplicidade do PROLOG quando comparado com as linguagens PLANNER reflecte as diferenças entre as origens. As linguagens PLANNER são extensões de linguagens de programação convencionais (sobretudo o LISP) e têm sido obtidas por enxerto das propriedades de nível mais alto sobre as de nível mais baixo pré-existentes. Assim, as propriedades de nível mais baixo, mais próximas da linguagem-máquina, embora tornadas redundantes pelas propriedades de nível mais alto permanecem nas novas linguagens. O PROLOG, pelo contrário, foi desenvolvido "ab início" a partir do outro extremo das linguagens de comunicação homem/máquina. Tem origem na lógica de predicados, e resulta da orientação para a mecanização da lógica e da sua automatização no computador. Além disso, a filosofia da programação através de lógica de predicados é a de providenciar serviços mais inteligentes através de interpretadores mais sofisticados ao invés de os providenciar através duma complexificação da linguagem de programação. Portanto, enquanto que as linguagens PLANNER cresceram pela elevação das linguagens-máquina a níveis mais próximos da comunicação com o homem, o PROLOG evoluiu como especialização da linguagem lógica humana até ao nível a que as máquinas o compreendem.

A escrita de programas PROLOG é próxima do modo lógico de abordar a resolução de problemas a resolver, não recorre a conceitos específicos da máquina, e por além do mais a lógica dos Predicados está dotada de um ingrediente de grande importância: a variável lógica. Em essência, a lógica dos predicados tem apenas esse único tipo de variável, cujo comportamento pode ser entendido independentemente do interpretador de lógica de predicados utilizado, ou da máquina subjacente. Em contraste, o comportamento das variáveis providenciada pelas linguagens PLANNER só pode ser entendido através do conhecimento de como o interpretador as trata. Para indicar isso, cada ocorrência de uma variável admite uma multiplicidade de prefixos (seis em CONNIVER), determinados por considerações de ordem implementacional (Pereira, 1975).

Um bom preceito, aliás, para escrever programas claros em PROLOG é o de efectuar primeiro a análise lógica do problema, sem começar de imediato a escrever o programa. Uma vez tornada clara a lógica do problema, o programa PROLOG para o resolver é basicamente a substanciação dessa lógica na linguagem do cálculo de predicados (evidentemente, haverá boas e más programações dum problema, boas e más análises da sua lógica).

É útil a propósito referir dois aspectos da especificação de algorítmos (van Emden, 1975): o aspecto imperativo, típico do nível de programação mais próximo da máquina (e.g. linguagem máquina, assemblers), e o aspecto descritivo, característico das linguagens de programação mais elevadas. Um programa em linguagem máquina soletra como o programa vai ser executado; contudo, é difícil perceber, sem explicações adicionais, o que está a ser computado. Este é o caso extremo duma especificação imperativa de um algorítmo. No outro extremo, numa especificação descritiva, apenas se descreve "o que" se pretende computar, sem indicar como fazê-lo. Pode pensar-se que é uma tarefa da programação automática o converter essa especificação numa especificação imperativa de comandos que dizem "o como" executar.

Estritamente falando, uma linguagem como o PL/1 ou como o ALGOL 60 é completamente imperativa: a cada "statement" correspondem comandos a serem executados. Contudo, o valor de tais linguagens reside no facto de que num programa bem escrito se torna possível entrever, mais facilmente no que nas linguagens próximas da máquina, o que está a ser feito: um tal programa tem um valor não só descritivo como imperativo. Alguns dos aspectos imperativos desapareceram do programa, tais como os detalhes relativos à alocação de memória e aos comandos envolvidos na chamada de um procedimento. A linguagem ABSYS (Elcock, 1968; Foster, 1968 e 1969) constitui uma interessante experiência, pois permite especificar algorítmos de modo mais descritivo. Nesse aspecto deve considerar-se como precursora das novas linguagens de programação que apareceram com a investigação em I.A. (Bobrow, 1974).

Na resolução de um problema é útil distinguir duas etapas. Primeiro há que formular o problema numa linguagem apropriada. Segundo, há que determinar o processo de procura de soluções para o problema, de acordo com a representação adoptada para o problema na linguagem que foi escolhida. Quanto mais estas duas etapas forem independentes, mais a definição da lógica do problema será distinta de especificação do controle necessário à busca de soluções. Não quer isto dizer que não haja representações melhores e piores da lógica dum problema, ou que para uma representação dada não existam especificações de controle mais e menos apropriadas. Por isso, o representar um problema envolve também saber olhá-lo de modo a simplificar o processo de encontrar-lhe soluções.

A Lógica dos Predicados é normalmente considerada como uma linguagem puramente descritiva, capaz quanto muito de exprimir o "que" há que ser feito por um programa, mas não o "como" fazê-lo. No entanto, relativamente a um procedimento de demonstração bem definido (tal como o do PROLOG), uma especificação lógica adquire um aspecto imperativo, que lhe permite assim especificar, simultaneamente com o que descreve, como utilizar isso que descreve. Isto é, diz não só "what is the case", mas ainda "how the case is".

Na linguagem PROLOG o aspecto descritivo nem sempre aparece com-

pletamente separado do aspecto imperativo, o qual define a estratégia ou controle de execução. Essa separação entre lógica e controle, entre os aspectos descritivos e imperativos duma linguagem, constitui um bom indicador para a caracterização de linguagens de nível mais alto: serão aquelas menos compremetidas com os aspectos imperativos dos algoritmos executados e que, sendo mais descritivas, facilitam não só a sua escrita como a sua leitura pelo ser humano. A Lógica dos Predicados tem uma origem priveligiada entre todas as linguagens de programação, pois resultou dos esforços de formalização dos raciocínios que uma linguagem natural permite explicitar. O resultado é uma linguagem suficientemente precisa para a comunicação com um computador, e no entanto bastante próxima do modo de expressão humana natural (Pereira, 1977).

Resta mencionar que não há nenhum mecanismo de controle de execução priveligiado a associar ao formalismo da lógica de resolução. As críticas que se possam fazer a sistemas dedutivos baseados na lógica dos predicados (e em particular no princípio da resolução) e utilizando este ou aquêle tipo de controle de execução, não podem ser entendidas como críticas à lógica de resolução de per se. Na verdade, estas críticas dirigem-se às estratégias referentes ao controle de execução, cujas falhas são afinal gerais, seja qual fôr o sistema dedutivo que as adopta.

4. CARACTERÍSTICAS PREDOMINANTES DO PROLOG

(1) Uma semântica declarativa herdada da lógica, além da semântica procedimental usual.
(2) Identidade de forma de programa e dados — as cláusulas podem ser empregues para expressar dados, e podem ser manipuladas como termos por interpretadores escritos em Prolog.
(3) Os argumentos de entrada e saída de um procedimento (grupo de cláusulas para o mesmo predicado) não precisam ser distinguidos como tais à partida, e o seu carácter pode variar de chamada para chamada.
(4) Os procedimentos podem ter múltiplos argumentos de saída bem como múltiplos argumentos de entrada.
(5) Os procedimentos podem gerar, por meio do retrocesso (backtracking), uma sequência de resultados alternativos. Trata-se de uma forma de iteração de alto-nível.
(6) Os termos providenciam estruturas gerais de registo (records) com qualquer número de campos. Um número ilimitado de tipos de registo pode ser utilizado, e não existem restrições para os tipos dos campos de um registo.
(7) A adequação de padrões (pattern matching) substitui o uso de selectores e constructores como meio de operar sobre dados estruturados.
(8) Estruturas de dados imcompletas (i.e. contendo variáveis livres) podem ser obtidas como resultado de um procedimento para mais tarde serem completadas por outros procedimentos, segundo uma ordem arbitrária.
(9) Toda a comunicação entre procedimentos concorrentes é assegurada pelas variáveis através da unificação, sem necessidade de interface explícito.
(10) O Prolog dispensa o *goto*, os loops *do*, *for* e *while*, o *assignment*, e as *references*.
(11) A semântica procedimental de um programa sintaticamente correcto é totalmente definida, não dando pois origem à execução de operações não definidas e portanto a comportamentos bizarros do programa.
(12) Nenhuma parcela do programa se ocupa com características da máquina ou implementação subjacentes.

NOVO!
ingrediente milagroso.
—A variável lógica.
1001 características
pelo preço de uma...

"Strings", "arrays"
procedimentos, listas,
base de dados,
"pattern-matching",
"semantic nets",
não determinismo,
representação de conhecimento,
"top-down", "bottom-up",
estruturas de controlo hierarquíco,
etc, etc,.

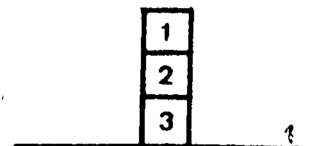
Tão fácil de usar!

PROLOG, a linguagem
de programação!

EXEMPLO: UTILIZAÇÃO DA LINGUAGEM PROLOG EM PROBLEMAS DE PLANEAMENTO.

ESTADO INICIAL

ESTADO OBJECTIVO



RESOLUÇÃO: INICIO,
MOVER 3 DE 1 PARA 4,
MOVER 2 DE 4 PARA 3,
MOVER 1 DE 4 PARA 2.

PROCEDIMENTO: :-PLANO(EM(1,2)&EM(2,3),INIÇIO).
TEMPO: 2.18 SEG.

referências bibliográficas

Prolog, uma linguagem de programação em lógica

PROLOG é uma linguagem baseada na lógica dos predicados (lógica de 1.ª ordem), criada em Marselha em 1970 pela equipa de Alain Colmerauer, e desenvolvida desde então em Marselha, Edimburgo e Lisboa. Existem actualmente versões do PROLOG, acessíveis em vários sistemas, dos quais destacamos:

VERSÕES DO PROLOG

SISTEMA	LINGUAGEM	TIPO	DATA	AUTOR	LOCAL
	ALGOL-W	interpretador	1970	P. Roussel	Marselha
	FORTRAN	interpretador	1972	Battani, Meloni	Marselha
IBM 370/158	VM/CMS	interpretador	1976	Roberts	Waterloo
Solar e Exorciser	ASSEMBLER	interpretador	1976	P. Roussel	Marselha
	PASCAL	interpretador	1976	Bruynooghe	Lovaina
ICL—1900	CDL	interpretador	1976	Szredi	Budapest
DECsystem-10	MACRO 10 e PROLOG	compilador e interpretador	1977	D. Warren, L. M. Pereira e F. Pereira	Edinburgh e Lisboa

NOTA: Todas as versões, exceptuando a 1.ª, encontram-se actualmente acessíveis.

O PROLOG é uma linguagem de muito alto nível, cuja importância poderá ser avaliada através da já vasta documentação publicada. Daí, a oportunidade de a apresentar aos leitores da revista.

Battani, G.; Meloni, H. (1973)
Interpreteur du langage de programmation PROLOG.
G. I. A., U. E. R., de Luming, Marseille, 1973.

Battani, G.; Meloni, H. (1974)
Un bel example de PROLOG en analyse et synthese.
G. I. A., U. E. R., de Luming, Marseille, Dec. 1974.

Battani, G.; Meloni, H. (1975)
Mise en oeuvre des contraintes phonologiques, syntaxiques et semantiques dans un systeme de comprehension automatique de la parole.
These de 3ème cycle, G. I. A., U. E. R. de Luming, Marseille, Juin 1975.

Bergman, M. (1973a)
Resolution par la demonstration automatique de quelques problemes en integration symbolique sur calculateur.
These de 3ème cycle, G. I. A., U. E. R. de Luming, Marseille, 1973.

Bergman, M.; Kanoui, H. (1973b)
Application of mechanical theorem proving to symbolic calculus.
G. I. A., U. E. R. de Luming, Marseille, 1973.

Bergman, M.; Kanoui, H. (1975)
SYCOPHANTE — systeme de calcul formal et d'integration symbolique sur ordinateur.
Rapport final, G. I. A., U. E. R. de Luming, Marseille, Oct. 1975.

Bobrow, D.; Raphael, B. (1974)
New programming languages for AI research.
Computing Surveys 6, 1974, 153-174.

Boyer, R. S.; Moore, J. (1972)
The sharing of structure in theorem-proving programs.
Machine Intelligence 7, Edinburgh, 1972, 101-116.

Bruynooghe, M. (1975)
The inheritance of links in a connection graph.
Report CW 2,
Applied Mathematics and Programming Division, Katholicke Universiteit Leuven, October 1975.

Bruynooghe, M. (1975)
An interpreter for predicate logic programs.
Part 1: Basic Principles.
Report CW 10,
Applied Mathematics and Programming Division, Katholicke Universiteit Leuven, October 1976.

Caneghan, M. van (1977)
Systeme d'analyse et de synthese morphologique en français pour l'exploitation de banques de donnees, tome 1, 2.
Univ. d'Aix — Marseille, 1977.

Chang, C. L.; Lee, R. C. T. (1973)
Symbolic logic and mechanical theorem proving.
Academic Press, 1973.

Clark, K. L.
Negation as failure.
Proceed. of the Workshop on Logic and Data Bases, Toulouse, 1977.

Clark, K. L.; Tärlund, S. A.
A first-order theory of data and programs.
Proc. IFIP 77.

Coelho, H. (1974)
An inquiry on the geometry machine and its extensions with a review of previous work.
Working report nº. 1 for INVOTAN, December 1974.
LNEC, 1976.

Coelho, H. (1977)
Natural language and data bases.
LNEC, DI, Novembro 1977.

Coelho, H.; Pereira L. M. (1976a)
GEOM: a PROLOG geometry theorem-prover.

Colmerauer, A. (1970)
Total precedence relations.
Journal of the ACM, vol. 17, nº. 1, Jan. 1970.

Colmerauer, A; Kanoui, H.; Pasero, R.; Roussel, P. (1973)
Un systeme de communication homme-machine en français.
G. I. A., U. E. R. de Luming, Marseille, 1973.

Colmerauer, A. (1973)
Les systemes-Q, ou un formalisme pour analyser et synthetiser des phrases sur ordinateur.
Pub. int. n.° 43, Depart. d'Informatique, Univ. de Montreal, 1973.

Colmerauer, A. (1974)
Programmation en langue naturelle.
G. I. A., U. E. R. de Luming, Marseille, 1974.

Colmerauer, A. (1975)
Grammaires de metamorphose.
G. I. A., U. E. R. de Luming, Marseille, 1975.

Colmerauer, A. (1976)
Lambda calculus and natural language.
Lecture notes, 1976 (not published).

Colmerauer, A. (1977)
Un sous-ensemble interessant du français.
Univ. d'Aix — Marseille, 1977.

Dahl, V.; Sambuc, R. (1976)
Un systeme de banque de donnees en logique du premir ordre, en vue de sa consultation en langue naturelle.
Univ. d'Aix — Marseille, 1976.

Dahl, V. (1977)
Un systeme deductif d'interrogation de banques de données en espagnol.
Univ. d'Aix — Marseille, 1977.

Davies, J. (1973)
POPLER 1.5 Refence Manual.
TPU Report 1, D. A. I., Edinburgh, 1973.

Deliyanni, A.; Kowalski, R. (1977)
Logic and semantic networks.
Depart. of computing and control, Imperial College, 1977.

Elcock, E. W. (1968)
Descriptions.
Machine Intelligence 3, B. Meltzer and D. Michie (eds.), Edinburgh Univ. Press, 1968, 173-179.

Emden, M. van (1973)
First-order predicate logic as a high level program language.
MIP-R-106 Depart. of Machine Intelligence, University of Edinburgh, 1973.

Emden, M. van; Kowalski, R. (1974)
The semantics of predicate logic as a programming language.
Report MIP-R-104, Depart. of Machine Intelligence, University of Edinburgh, 1974.

Emden, M. van (1975)
Programming with Resolution Logic.
Research report CS-75-30.
University of Waterloo, November 1975.

Emden, M. van (1977)
Computation and deductive information retrieval.
Research report CS-77-16, May 1977.

Feldman, L. A. et Al. (1972)
Recent developments in SAIL — an ALGOL based language for AI.
FJCC, 1972.

Foster, J. M. (1968)
Assertions: programs written without specifying unnecessary order.
Machine Intelligence 3, B. Meltzer and D. Michie (eds), Edinburgh University Press, 1963, 387-391.

Foster, J. M.; Elcock, E. W. (1969)
Absys 1: an incremental compiler for assertions.
Machine Intelligence 4, B. Meltzer and D. Michie (eds.), Edinburgh University Press, 1968, 387-391.

Guizol, J. (1976)
Synthese du français a partir d'une representation et logique du premier ordre.
Univ. d'Aix — Marseille, 1976.

Hewitt, C. (1973)
Description and theoretical analysis of PLANNER.
MIT, AI Memo 251, 1972.

Hewitt, C.; Smith, B. (1974)
Towards a programming apprentice.
Proceed. of AISB Conference, 186-213, 1974.

Hill, R. (1974)
LUSH resolution and its completness.
DCL Memo 78, Depart. of Artificial Intelligence, University of Edinburgh, 1974.

Hopcroft; Ullman (1969)
Formal languages and their relation to automata.
Addison — Wesley, 1969.

Jourbert, M. (1974)
Un systeme de resolution de problems a tendance naturelle.
These de 3ème cycle, G. I. A., U. E. R. de Luming, Marseille, 1974.

Kanoui, H. (1973)
Application de la demonstration automatique aux manipulations algebriques et a l'integration formelle sur ordinateur.
G. I. A., U. E. R. de Luming, Marseille, 1973.

Kanoui, H. (1975)
Some aspects of symbolic integration via predicate logic programming.
Paper in Proc. of IJCAI-4, Tiblisi, 1975.

Kowalski, R.; Kuehner, D. (1971)
Linear resolution with selection function.
Artificial Intelligence 2, 1971, 227-260.

Kowalski, R. (1974)
Predicate logic as a programming language.
Proc. IFIP 74.

Kowalski R. (1974)
Logic for problem-solving.
Memo 75, Dept. of Computational Logic, Univ. of Edinburgh, 1974.

Kowalski, R. (1976)
Logic and data bases.
Dept. of Computation and Control.
Imperial College, 1976.

Kowalski, R. (1977)
Algorithm = Logic + Control.
Technical report, Depart. of Computation and Control, Imperial College 1977.

Kowalski, R. (1977)
General laws in Data Description.
Depart. of Computing and Control, Imperial College, 1977.

Lehn, K. A. van (1973)
Sail user manual.
Standford A. I. Laboratory.
Memo no. 204, July 1973.

McDermott, D. V.; Sussman, G. J. (1973)
The CONNIVER reference manual.
MIT, AI memo 259, 1972.

Meloni, H. (1976)
PROLOG, mise en route de l'interpreteur et exercices.
Groupe d'Intelligence Artificielle, Univ. d'Aix — Marseille II, 1976.

Moore, J. S. (1973)
Computational logic: Structure sharing and proof of program properties.
Memo no. 68, Depart. of Artificial Intelligence, 1973.
University of Edinburgh, 1973.

Pasero, R. (1973)
Represntation du Français en logique du premier ordre en vue de dialoguer avec un ordinateur.
G. I. A., U. E. R. de Luming, Marseille, 1973.

Pasero, R. (1976)
Un essai de communication sensee en langue naturelle.
Univ. d'Aix — Marseille, 1976.

Pereira, L. M.; Meltzer, B. (1975)
Implementatioin of an efficient predicate logic interpreter based on Earley deduction.
Scientific proposal report for SRC, April 1975.

Pereira, L. M.
Users's Guide to DECsystem-10 Prolog.
LNEC, DI, 1977.

Pereira, L. M. (1977)
PROLOG, uma linguagem de programação em lógica.
LNEC, DI, Maio 1977.

Reboh, R.; Sacerdoti, E. (1973)
A preliminary QLISP manual.
SRI AI Center, Techn. note 81, 1973.

Reiter, R. (1972)
The use of models in automatic theorem-proving.
Univ. of British Columbia, Techn. report 72-04, Sept. 1972.

Roberts, G. M. (1977)
An implementation of PROLOG.
Msc. thesis.
University of Waterloo, Ontario, 1977.

Roussel, P. (1972)
Definition et traitement de l'egalité formelle on demonstration automatique.
G. I. A., U. E. R. de Luming, Université d'Aix — Marseille, 1972.

Roussel, P. (1975)
PROLOG — Manuel de reference et d'utilization.
G. I. A., U. E. R. de Luming, Unid'Aix — Marseille, 1975.

Rulifson, J. F. et Al. (1973)
QA-4: a procedural calculus for intuitive reasoning.
SRI AI Center, Techn. memo 73, 1973.

Salomaa, A. (1973)
Formal languages.
Academic Press, 1973.

Sussman, G. J.. Winograd, T. (1970)
MICRO - PLANNER référence manual.
MIT AI Memo 203, 1970.

Sussman, G. J. (1972)
Why conniving is better than planning.
MIT AI Memo 255A, 1972.

Teitelman, W. (1974)
INTERLISP refence manual.
Xerox Palo Alto, Research Center, Oct. 1974.

Wall, R. E. (1972)
Introduction to mathematical linguistics.
Prentice Hall, 1972.

Warren, D. (1974a)
Warplan: a system for generating plans..
DAI, Univ, of Edinburgh, memo no. 76, June 1974.

Warren, D. (1974b)
Lecture notes on PROLOG.
D. A. I., University of Edinburgh, Nov. 1974.

Warren, D. (1975a)
Examples for Warplan and PROLOG: the car assembly problem and two scene analysis problems.
D. A. I., University of Edinburgh, 1975.

Warren, D. (1975b)
Epilog: an user's guide to DEC-10 PROLOG.
DAI, report stored in 400,400 area of DEC-10 of the Univ. of Edinburgh, August 1975.

Warren, D. (1975c)
User's guide to PROLOG superviser SVW.
D. A. I., University of Edinburgh, Nov. 1975.

Warren, D. (1976a)
Generating conditional plans and programs.
D. A. I., University of Edinburgh, 1976.

Warren, D. (1976b)
Introduction notes on PROLOG implementation.
Working paper, Depart. of Artificial Intelligence, University of Edinburgh, 1976.

Warren, D. (1977a)
Compiler writing and logic programming.
D. A. I., Edinburgh, 1977.

Warren, D. (1977b)
Implementing PROLOG — compiling predicate logic programs. Vol. 1 & 2.
DAI research report nº. 39, May 1977.

Warren, D.; Pereira, L. M.; Pereira, F. (1977)
PROLOG — The language and its implementation compared with LISP.
LNEC, DI, 1977.

Welham, R. (1976)
Geometry problem solving.
D. A. I research report no. 14, January 1976.

ESPAÇO U.N.L.

INTRODUÇÃO

À

INVESTIGAÇÃO

OUTUBRO
NOVEMBRO

Serviços Académicos

Universidade Nova de Lisboa

Quinta do Cabeço
Seminário dos Olivais
Lisboa - 6