1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-4211-PRE/1092		*Julho, 1987*	☐ Interna ☒ Externa ☐ Restrita
4. Origem: *DPI*	Programa: *ATDPI*		

6. Palavras chaves - selecionadas pelo(s) autor(es)

ENGENHARIA DE SOFTWARE
APL
HISTÓRICO
SEGURANÇA DE FUNCIONAMENTO
EDIÇÃO AUTOMÁTICA DE PROGRAMA

7. C.D.U.: *681.3.06*

8. Título *INPE-4211-PRE/1092*

NOÇÃO DE HISTÓRICO DE UMA VARIÁVEL E APLICAÇÃO À PROGRAMAÇÃO PELO EXEMPLO

10. Páginas: *11*

11. Última página: *11*

12. Revisada por

Flávio Roberto D.Velasco

9. Autoria

Gerald Jean Francis Banon

Assinatura responsável

13. Autorizada por

Dr. Marco Antonio Raupp
Diretor Geral

14. Resumo/Notas

Neste trabalho é introduzida uma definição de histórico de uma variável e é mostrado como essa noção pode servir de base a uma nova técnica de programação, chamada de programação pelo exemplo, que dispensa a fase de edição de programa na sua forma tradicional e garante a síntese de programa com boa segurança de funcionamento. Essa técnica é ilustrada através de um exemplo de criação de funções no ambiente APL.

15. Observações *Trabalho apresentado no III Encontro Regional de Matemática Aplicada e Computacional, realizado de 18 a 20 de Maio de 1987, INPE, São José dos Campos, SP.*

NOÇÃO DE HISTÓRICO DE UMA VARIÁVEL E APLICAÇÃO À PROGRAMAÇÃO PELO EXEMPLO

Gerald Jean Francis Banon*

SUMÁRIO

Neste trabalho é introduzida uma definição de histórico de uma variável e é mostrado como essa noção pode servir de base a uma nova técnica de programação, chamada de programação pelo exemplo, que dispensa a fase de edição de programa na sua forma tradicional e garante a síntese de programa com boa segurança de funcionamento. Essa técnica é ilustrada através de um exemplo de criação de funções no ambiente APL.

INTRODUÇÃO

O objetivo deste trabalho é introduzir algumas idéias que podem ser, uma vez implementadas, de grande utilidade como ferramenta para ajudar no desenvolvimento de novas funções (i.e. programas) na área de computação.

Nesta área uma questão ainda em aberto é como resolver de maneira produtiva o problema da programação. Inúmeros esforços já foram empregados neste sentido levando às mais variadas propostas.

Assim, foram introduzidas as noções de programação estruturada que consistem em regras a serem seguidas pelos programadores que usam linguagens tradicionais como FORTRAN, PLI, PASCAL, BASIC. Hoje em dia, continuando nesta linha, são oferecidos editores com novos comandos permitindo a escritura estruturada. A chegada dos sistemas especialistas trouxe de uma certa forma uma solução totalmente nova e original pois neste contexto a tarefa de programação é feita pela própria máquina. Uma vez implementado seu motor de inferência e armazenada sua base de conhecimento, é o próprio sistema especialista que estabelece, na medida do possível, a sequência dos passos que levarão à solução.

A abordagem considerada aqui situa-se entre os dois extremos mencionados acima. Neste trabalho, o problema da programação ainda é deixado ao progra

---

*Pesquisador Senior do Departamento de Processamento de Imagens do Instituto de Pesquisas Espaciais.

mador, mas ele o resolve usando uma linguagem não convencional como é o caso do APL considerado aqui.

O primeiro aspecto original do APL, como linguagem funcional, está na possibilidade de manipular com grande facilidade a estrutura dos dados (e.g. transformar uma matriz em vetor) e de executar operações independentemente dessa estrutura (e.g. a função "+" realiza tanto a soma de vetores como de matrizes). Com essa possibilidade em muitas aplicações os programas podem ser escritas de uma maneira linear ou seja sem nenhuma necessidade de recorrer explicitamente aos tradicionais "GOTO", "IF" ou "DO" do FORTRAN por exemplo.

Em APL para somar 1 a cada elemento de um vetor numérico A de tamanho qualquer basta escrever:

```
1 + A
```

(não há necessidade de usar explicitamente nenhum "DO").

Para somar 1 a cada elemento positivo de um vetor numérico A de tamanho qualquer basta escrever:

```
(A > 0) + A
```

(não há necessidade de usar explicitamente nenhum "IF" e nenhum "GOTO").

O segundo aspecto interessante do APL é seu uso em modo interativo que oferece uma grande flexibilidade de manuseio dos dados. A todo momento é possível mandar executar novas tarefas (tarefas descritas em uma linha), criar novas variáveis e testá-las imediatamente. Por exemplo, para criar a variável B resultado do cálculo acima basta escrever:

```
B ← (A > 0) + A.
```

Para testar se todos os elementos de B são maiores ou iguais aos de A basta escrever:

```
∧/B ≥ A
```

e observar o resultado (se a asserção acima for verdadeira então o resultado será 1 senão será 0). Esses dois aspectos, funcional e interativo, do APL são extensivamente aproveitados neste trabalho. Num primeiro tempo a noção de histórico de uma variável (como A e B acima) é introduzida com o objetivo de documentar o trabalho do programador e economizar espaço em memória. Num segundo tempo é mostrado como, com a transformação desses históricos em funções executáveis (i.e. programas), o programador dispõe de uma nova técnica de programação chamada aqui de programação pelo exemplo. Nesta técnica os históricos são vistos como exemplos de resolução de problemas com dados

reais e é a partir destes exemplos que se programam novas funções (i.e. programas).

Usando essa técnica abandona-se o hábito, vindo das linguagens compiladas, de separar o trabalho de programação em edição/execução. Programar pelo exemplo significa resolver passo a passo interativamente um problema com dados reais, o sistema de gerenciamento guardando esses passos na forma de históricos, e quando o problema for resolvido criar automaticamente, a partir do histórico da variável resultado, uma nova função (i.e. programa). Em outros termos elimina-se a fase de edição na sua forma tradicional. Quando se programa pelo exemplo, só se pode prosseguir na resolução de um problema depois de ter obtido todos os resultados intermediários antecessores. Dessa forma pode-se garantir que a nova função (i.e. programa), além de não comportar erro de sintaxe, funciona sem erro, pelo menos quando são aplicados os dados originais usados no exemplo.

Finalmente é interessante notar que as idéias apresentadas aqui surgiram dentro de processamento de imagens de satélite por computador [1]. Uma das preocupações em processamento de imagens é a melhoria do visual das imagens. Essa melhoria não segue necessariamente critérios objetivos e muitas vezes é o resultado de uma série de experiências. Daí surgiu a noção de histórico de uma variável. A variável contém os dados da imagem a ser visualizada e seu histórico indica as condições nas quais ela foi obtida. Mas sendo geral essa noção poder ser apresentada e usada sem referência feita a um domínio particular de aplicação.

NOÇÃO DE HISTÓRICO DE UMA VARIÁVEL

Neste trabalho uma variável é simplesmente o nome dada a uma constante ou ao resultado de uma computação. Nas expressões, chamadas aqui também de linhas:

```
A ← 1 + 3
B ← 2 x A
```

A e B são variáveis.

Partindo da facilidade oferecida pelo APL de programar de maneira linear (c.f. introdução), pode-se associar a cada variável seu histórico que é definido da seguinte maneira: o histórico de uma variável é a sequência das linhas (e somente essas) que foram usadas (implicitamente* ou explicitamente)

---

*A palavra implicitamente refere-se às linhas definindo os valores de "default" usados por certas funções (i.e. programas).

para criar a variável e que quando executada (de cima para baixo) reproduz o conteúdo da variável.

Assim o histórico de A (ver acima) comporta uma linha:

A ← 1 + 2

o de B comporta duas linhas:

A ← 1 + 2
B ← 2 x A.

Na prática é o próprio sistema gerenciador que se encarrega de criar e armazenar o histórico de cada nova variável.

A segunda parte da definição do histórico implica que todas as funções aparecendo no histórico tem que depender somente das variáveis aparacendo no histórico. Por exemplo o resultado do cálculo de uma função (i.e. de um programa) não pode depender de uma variável externa como o "tempo", senão não seria possível reproduzir este resultado numa outra execução.

Em certos aspectos esta definição é restritiva, mas assim ela garante que um histórico será sempre executável sem erro. Ela oferece também a possibilidade muito interessante para certas aplicações de economizar espaço em memória. Uma vez salvo o histórico de uma variável (este geralmente ocupa pouco espaço em memória), esta pode ser apagada liberando espaço em memória e recalculada a partir do seu histórico quando for preciso.

Um outro aspecto interessante dos históricos é que eles podem servir para documentar o resultado de um cálculo uma vez este armazenado sob o nome de uma variável. Nos históricos aparecem todos os dados (nome das funções e valores dos seus argumentos) que levaram àquele resultado (ver figura 1).

Finalmente, um último aspecto valioso dos históricos é sua aplicação potencial à programação pelo exemplo.

PROGRAMAÇÃO PELO EXEMPLO

Como foi dito na introdução, um histórico pode ser visto como um exemplo de resolução de um problema com dados reais. Na medida em que o sistema gerenciador encarrega-se de construir os históricos no momento em que estão criadas as variáveis, o programador tem à sua disposição uma bateria de exemplos de resolução de problemas com dados reais. Quando for preciso, estes exemplos podem servir como base para resolver os mesmos problemas mas com dados reais diferentes. Nesta situação é conveniente transformar o histórico em uma função (i.e. programa). Esta transformação pode ser deixada a cargo do

```
NOME←'ENI DATA'
A←NOME EI 1 402 83 512
⎕CT←1E¯13
⎕IO←1
A75←¯1+(0.75×(DISTRIBUICAO A))⍳1    a: A75←168
VAR6←A75[A
```

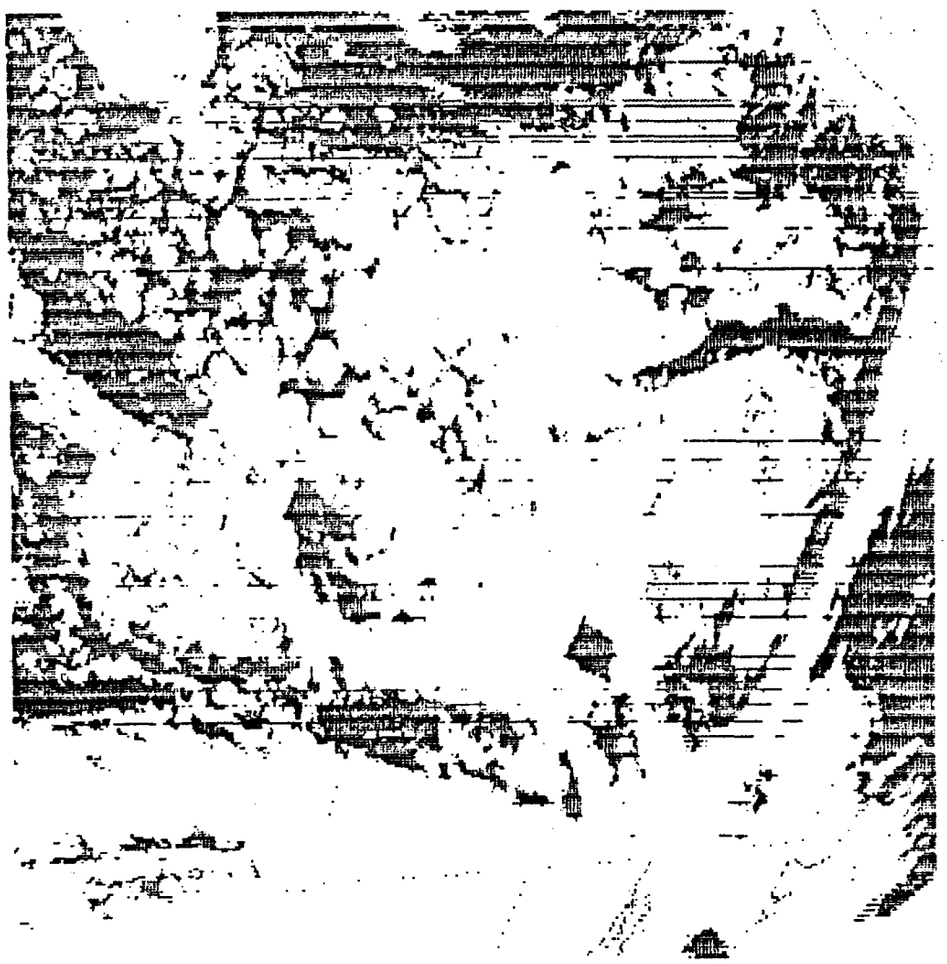


Fig. 1 - O histórico serve para documentar um resultado.
Acima o histórico da variável VAR6.
Abaixo a visualização de VAR6 (VAR6 é uma imagem binária).
No histórico EI e DISTRIBUIÇÃO são nomes de duas funções.

próprio gerenciador. Os exemplos armazenados em forma de histórico sendo sem falha de funcionamento, as funções transformadas destes históricos serão também seguras para um certo conjunto de dados reais que deverá ser identificado.

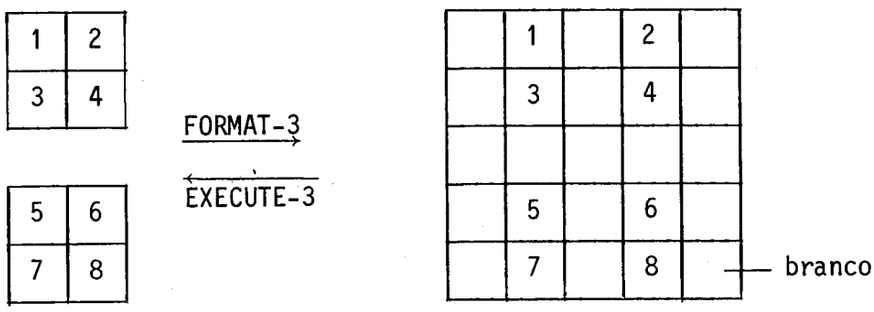
Assim, a técnica descrita acima, e chamada de programação por exemplo, aparece como uma ajuda à programação que conduz, sem passar pela fase tradicional da edição, a criação de funções (i.e. programas) sem falha de funcionamento.

A fim de testar no ambiente APL as idéias apresentadas aqui, foi criada uma nova função sistema dentro do conjunto das funções sistemas já existentes na linguagem APL2 (Release 1 ou 2) [2]. A descrição completa do uso dessa função é feita em [3]. Ela é denotada ⎕H e tem as seguintes funções:

- produzir o histórico de uma variável já em memória;
- criar em memória uma variável a partir do seu histórico (função dual da anterior);
- criar em memória uma nova função a partir do histórico de uma variável já em memória.

Para ilustrar a técnica de programação pelo exemplo através do uso da função ⎕H, uma curta sessão APL é apresentada em seguida. Nessa sessão são criadas duas funções duais (uma inversa da outra) chamadas FORMAT-3 e EXECUTE-3.

O problema é transformar uma variável numérica com três dimensões em uma variável alfanumérica com duas dimensões e vice-versa como mostrado no exemplo da figura 2.



variável numérica
(3 dimensões: 2 blocos
de 2 linhas
de 2 colunas)

variável alfanumérica
(2 dimensões: 5 linhas
de 5 colunas)

Fig. 2 - Exemplo de transformação de dados

Em seguida são descritos todos os passos da sessão APL.

- Passos relativos à criação da função FORMAT-3

1) Criação da variável numérica R contendo os dados a serem transformados.

2) Visualização de R.

3) Criação da variável alfanumérica Z transformada de R. Uma só linha APL é necessária.

4) Visualização de Z.

5) Visualização das dimensões de Z. A estrutura de Z é conforme aquilo desejado.

6) Criação da função FORMAT-3 usando ⎕H.

7) Execução de FORMAT-3 usando os dados originais contidos em R. O resultado é conforme o esperado.

8) Criação da variável R contendo o resultado da execução de FORMAT-3 usando outros dados, e visualização da sua dimensão.

9) Visualização de R.

10) Visualização da função FORMAT-3. As linhas 2 e 3 são linhas de comentários criados automaticamente a partir do histórico de Z e servem como exemplo de uso da função. As linhas 4 e 5 servem a definir o valor de "default" das variáveis sistemas usadas pelas funções aparecendo na linha 6.

11) Visualização do histórico de Z. A função FORMAT-3 foi criada a partir deste histórico.

- Passos relativos à criação da função EXECUTE-3

É preciso recriar as três dimensões a partir da forma alfanumérica definida no passo 8, i.e. 212. Para isto são criadas duas variáveis A1 e A2.

12) Criação da variável A1 contendo o vetor de linhas de R.

13) Deteção das linhas brancas de R. A linha do meio é branca.

14) Criação da variável A2 contendo o resultado do passo 13.

15) Determinação da 1ª dimensão (número de blocos).

16) Determinação da 2ª dimensão (número de linhas).

17) Visualização da dimensão da primeira linha de R.

18) Determinação da 3ª dimensão (número de colunas).

19) Estudo do caso onde as linhas comportem um só número. Neste caso o algoritmo do passo 18 produz o vazio, o que não é correto.

20) Verificação que o algoritmo correto tem que comportar uma vetorização.

21) Determinação da 3ª dimensão (caso geral).

22) Reconstituição das 3 dimensões.

23) Criação da variável numérica Z transformada de R.

24) Visualização das dimensões de Z. A estrutura de Z é conforme ao esperado.

25) Visualização de Z. O resultado é conforme ao esperado.

26) Criação da função EXECUTE-3 usando ⎕H.

27) Visualização da função EXECUTE-3.

28) Criação de uma nova variável numérica R.

29) Verificação da dualidade das funções FORMAT-3 e EXECUTE-3.

```
1)      R←2 2 2⍴⍳8

2)      R
  1 2
  3 4

  5 6
  7 8

3)      Z←⍉,[1.1]⊂[2 3]R

4)      Z
   1 2
   3 4

   5 6
   7 8

5)      ⍴Z
  5 5

6)      1 ⎕FX 'Z←FORMAT_3 R'
  FORMAT_3
```

```
 7)       ⍴FORMAT_3 R
   5 5

 8)       ⍴R←FORMAT_3 2 1 2⍴⍳4
   3 5

 9)       R
    1 2

    3 4

10)       ⎕CR 'FORMAT_3'
   Z←FORMAT_3 R;⎕IO
   ⍝: ⎕IO←1
   ⍝: R←2 2 2⍴⍳8    ⍝: R←2 2 2⍴1 2 3 4 5 6 7 8
   ⎕IO←1
   ⎕PP←10
   Z←↑,[1.1]⊂[2 3]R

11)       1 ⎕CR 'Z'
   ⎕IO←1
   R←2 2 2⍴⍳8    ⍝: R←2 2 2⍴1 2 3 4 5 6 7 8
   ⎕PP←10
   Z←↑,[1.1]⊂[2 3]R
```

```
12)       A1←⊂[2]R

          ∧/¨' '=A1
13) 0 1 0

14)       A2←∧/¨' '=A1

15)       1++/A2
   2

16)       +/∧\~A2
   1

17)       ⍴↑A1
   5

18)       ⍴⍪↑A1
   2

19)       ⍴⍪'1'

20)       ⍴,⍪'1'
   1
```

```
21)       ⍴,⍤1↑A1
   2
22)       (1++/A2),(+/∧\~A2),⍴,⍤1↑A1
   2 1 2

23)       Z←((1++/A2),(+/∧\~A2),⍴,⍤1↑A1)⍴⍤1∈A1

24)       ⍴Z
   2 1 2

25)       Z
   1 2

   3 4


26)       1 ⎕OH 'Z←EXECUTE_3 R;'
   EXECUTE_3


27)       ⎕CR 'EXECUTE_3'
   Z←EXECUTE_3 R;⎕CT;⎕IO;A1;A2;⎕H
   ⍝: ⎕IO←1
   ⍝: R←FORMAT_3 2 1 2⍴⍳4
   ⎕IO←1
   A1←⊂[2]R          ⍝: A1←' 1 2 ' '       ' ' 3 4 '
   ⎕CT←1E¯13
   A2←∧/¨' '≠A1      ⍝: A2←0 1 0
   Z←((1++/A2),(+/∧\~A2),⍴,⍤1↑A1)⍴⍤1∈A1

28)       R←3 4 5⍴⍳60

29)       R≡EXECUTE_3 FORMAT_3 R
   1
```

CONCLUSÃO

Foi visto como a noção do histórico de uma variável pode servir de base a uma nova técnica de programação, chamada de programação pelo exemplo. Essa técnica dispensa a fase tradicional da edição e garante o máximo de segurança de funcionamento.

Neste trabalho não foi abordado a fundo o problema da integridade dos históricos e como seria possível garantir essa integridade. Algumas soluções neste sentido são dadas em [3]. Não foi também considerada a possibilidade de desenvolver uma programação pelo exemplo baseada numa definição menos restritiva de histórico. Esses dois pontos poderiam ser objeto de pesquisas futuras.

REFERÊNCIAS

[1] BANON, GERALD J.F. *Implementação de um Sistema de Tratamento de Imagens Usando uma Definição Ampla para Imagem.* Anais do III Simpósio Brasileiro de Sensoriamento Remoto, Rio de Janeiro, 28-30 de novembro de 1984.

[2] APL2 Programming: Language Reference manual IBM nº SH20-9227-0, 1985.

[3] BANON, GERALD J.F. *A Service for APL2 Global Variable Manipulation.* Relatório Técnico nº CCB 038 - Centro Científico IBM - Brasil, maio 1986.