PROCEDIMENTO, FUNÇÃO, OBJETO OU LÓGICA? LINGUAGENS DE PROGRAMAÇÃO VISTAS PELOS SEUS PARADIGMAS

Maria Cecília Calani Baranauskas*

INTRODUÇÃO

A escolha da linguagem de programação para uma aplicação específica, principalmente no caso de contextos educacionais, requer uma atenção particular ao paradigma subjacente a linguagem. O objetivo deste artigo é situar algumas linguagens de programação usadas em Educação, principalmente Logo e Prolog, entre os principais paradigmas existentes, sob a ótica de "meios" diferentes onde problemas são representados e resolvidos. Esta abordagem às linguagens pelos seus paradigmas leva a uma reflexão sobre metodologias de uso de linguagens de programação no contexto educacional.

Várias definições podem ser encontradas na literatura, para "paradigma de programação". Papert, referindo-se a linguagens suportadas por novas arquiteturas, define paradigma de programação como um "quadro estruturador1" que é subjacente à atividade de programar e coloca que a escolha do paradigma de programação pode mudar notavelmente "*a maneira como o programador pensa sobre a tarefa de programar*" (Papert, 1991, p.8). De acordo com a visão proposta neste artigo, esse quadro estruturador já existe no nível de abstração que linguagens como Lisp e Prolog, por exemplo, propuseram sobre a arquitetura von Neumann.

Recuperando o contexto histórico da evolução das linguagens de programação, pode-se dizer que elas representam graus variados de abstração da arquitetura subjacente, chamada von Neumann.

Conhecer as origens dos paradigmas de programação envolve conhecer um pouco da história da evolução das linguagens de programação. Os computadores disponíveis no final da década de 40 e início da década de 50, além dos problemas decorrentes da tecnologia da época, eram difíceis de serem programados pela ausência de *software*. Na falta de linguagens de programação de alto nível, ou mesmo linguagens de montagem, a programação era feita em código de máquina (por exemplo, uma instrução para "somar", deveria ser especificada por um código em vez do seu uso textual). Essa maneira de programar tornava os programas ilegíveis, além de ser bastante complicado o seu processo de depuração. Do ponto de vista do programador, essa foi uma motivação importante para a criação das linguagens de montagem e seus montadores. Além disso, as aplicações numéricas da época requeriam o uso de certas facilidades que não estavam incluidas no *hardware* das máquinas de então, (números reais, acesso a elementos de um

*Depto. de Ciências da Computação - IMECC - UNICAMP
Núcleo de Informática Aplicada à Educação - NIED

conjunto por seu índice, por exemplo) surgindo daí a criação de linguagens de mais alto nível que incluissem tais recursos. O paradigma "procedural", é o que mais se aproxima do uso da arquitetura von Neumann como modelo para representação da solução de um problema a ser resolvido pela máquina. Segundo o paradigma procedural, programar o computador significa "dar-lhe ordens" que são executadas sequencialmente. Em tal paradigma, "representar" a solução de um problema para ser resolvido pelo computador envolve escrever uma série de ações (procedimentos) que, se executadas sequencialmente, levam à solução. Ou seja, o programa representa a prescrição da solução para o problema. As linguagens de programação procedurais (ou imperativas) como por exemplo Fortran, Pascal, C, Módula-2, formam a maior classe das linguagens existentes até então. Várias razões poderiam ser usadas para explicar o crescimento da classe de linguagens procedurais; devemos apontar o papel histórico do uso do computador em aplicações numéricas e o uso do computador dentro do próprio domínio da ciência da computação. O desenvolvimento de tais linguagens tem sido feito por especialistas em computação, para uso de especialistas em computação, dentro do seu próprio domínio, onde questões de eficiência e desempenho são fundamentais. A classe das linguagens procedurais continuará a evoluir, enquanto a arquitetura subjacente dos computadores for a arquitetura von Neumann.

O paradigma funcional de programação surgiu com o desenvolvimento da linguagem Lisp (de List Processing) por John McCarthy em 1958. Lisp foi projetada, portanto, numa época em que só existia processamento numérico, para atender aos interesses dos grupos de Inteligência Artificial no processamento de dados simbólicos. Apesar do estilo imperativo de programar ser bem aceito entre programadores, provavelmente em função do tipo de aplicações realizadas na época, o vínculo da linguagem com a arquitetura von Neumann, excetuando-se as razões técnicas de eficiência, do ponto de vista de metodologia de desenvolvimento de programas, apresenta-se como uma restrição desnecessária (Sebesta, 1988). Surgiu como uma nova base para projeto de linguagens, o uso de funções matemáticas e composição de funções, introduzindo um novo modelo para representação do problema a ser resolvido pela máquina. Segundo o paradigma funcional, programar o computador significa definir funções, aplicar funções e conhecer o comportamento de funções na máquina; os mecanismos de controle, no programa, passam de iterativos a recursivos. Assim, representar a solução de um problema para ser resolvido num ambiente funcional passa a necessitar de uma abordagem completamente diferente dos métodos usados em linguagens imperativas.

Programação orientada a objetos é um novo paradigma, que surgiu em paralelo à criação de uma linguagem de programação, chamada Smalltalk, proposta por Alan Kay em 1972. A idéia básica do paradigma orientado a objetos é imaginar que programas simulam o mundo real: um mundo povoado de objetos. Dessa maneira, linguagens baseadas nos conceitos de simulação do mundo real devem incluir um modelo de objetos que possam enviar e receber mensagens e reagir a mensagens recebidas. Esse conceito é baseado na idéia de que no mundo real frequentemente usamos objetos sem precisarmos conhecer como eles realmente funcionam. Assim, programação orientada a objetos fornece um ambiente onde múltiplos objetos podem coexistir e trocar mensagens entre si.

Programação em lógica é uma teoria que representa um modelo abstrato de computação sem relação direta com o modelo von Neumann de máquina. Prolog (de Programming in Logic), surgiu no início dos anos 70, dos esforços de Robert Kowalski, Maarten van Emden e Alain

Colmerauer e é a linguagem de programação desenvolvida em máquina sequencial, que mais se aproxima do modelo de computação de programação em lógica. O enfoque do paradigma da programação em lógica para se representar um problema a ser resolvido no computador, consiste em expressar o problema na forma de lógica simbólica. Um processo de inferência é usado pela máquina para produzir resultados. Segundo o modelo de programar proposto por Prolog, o significado de um "programa" não é mais dado por uma sucessão de operações elementares que o computador supostamente realiza, mas por uma base de conhecimento a respeito de certo domínio e por perguntas feitas a essa base de conhecimento, independentemente. Dessa maneira, Prolog pode ser visto como um formalismo para representar conhecimento a respeito do problema que se quer resolver, de forma declarativa (descritiva). Existe, por trás do programa uma máquina de inferência, em princípio "escondida" do programador, responsável por "encontrar soluções" para o problema descrito.

Programar nos diferentes paradigmas significa, portanto, representar, segundo modelos diferentes, a solução do problema a ser resolvido na máquina. Cada linguagem que suporta determinado paradigma representa, portanto, um "meio" onde o problema é "resolvido". Enquanto "meio de expressão" e de "comunicação" com a máquina, a linguagem e, indiretamente o seu paradigma, "moldam" a representação do problema a ser resolvido. Assim, na atividade de programar, mudar de paradigma significa muito mais do que conhecer as entidades sintáticas e semânticas da nova linguagem, o processo de pensamento também deve ser mudado, ajustando-se ao novo meio de representação do problema. O psicolinguista Benjamin Lee Whorf (citado em Johanson (1988)) em seu trabalho com linguagens naturais sugere que existem estruturas inerentes à linguagem que falamos das quais não nos damos conta, mas que têm uma profunda influência em nossos pensamentos. Ele argumenta que os padrões disponíveis nas estruturas das linguagens que usamos são mais importantes que as palavras em si. O trabalho de Whorf, embora não se refira a linguagens artificiais, concorda com o conceito de Papert sobre paradigma de programação como uma estrutura subjacente à linguagem de programação que influencia a maneira como o sujeito encara a tarefa de programar e faz-nos questionar a respeito do tipo de pensamentos que um determinado paradigma desperta, se considerarmos as implicações cognitivas do uso de linguagens de programação em resolução de problemas.

Neste artigo a palavra "paradigma" é usada, no contexto de linguagens de programação, para representar diferentes "modelos" de representação do problema a ser resolvido na máquina. Em minha ótica, o entendimento de tais modelos é fundamental no *design* de metodologias para desenvolvimento de programas em uma dada linguagem; na criação de ambientes de aprendizado baseado no computador e também no ensino/aprendizado de linguagens de programação de modo geral. Assim, um dos objetivos deste artigo é mostrar que os paradigmas de programação fornecem diferentes "meios" para representação de problemas. Outro objetivo é mostrar que algumas linguagens (como Logo e Prolog) englobam vários desses "meios" e a mudança de um meio para outro não é um processo suave para usuários não sofisticados.

Para situar os paradigmas de programação como "meios" diferentes onde problemas são representados, procuraremos exemplificá-los através da representação de um mesmo problema, nesses diferentes meios. Apesar de cada paradigma definir uma classe de problemas à qual as linguagens melhor se adequam, as linguagens de programação de alto nível são consideradas "de propósito geral". Assim sendo, correndo o risco de sermos "parciais" a esse respeito, escolhemos como o "problema" a ser exemplificado nos vários paradigmas, um clássico na

literatura de computação: o problema de calcular o fatorial de determinado número. Nas seções seguintes o uso desses quatro paradigmas será discutido no contexto de duas linguagens de programação muito citadas no contexto educacional: Logo e Prolog. Alguns *bugs* de novatos no processo de aprender essas linguagens serão mostrados para ilustrar a problemática da não observância dos paradigmas subjacentes.

OS "MEIOS" GERADOS PELOS PARADIGMAS DE PROGRAMAÇÃO

Os paradigmas das linguagens de programação, interpretados como "meios" onde problemas são resolvidos, apresentam diferentes significados para "programa" e para a "máquina que executa o programa". Consequentemente, tem-se diferentes maneiras de pensar e representar problemas, conforme será ilustrado a seguir.

O meio procedural

O meio procedural pretende "imitar" a máquina von Neumann; o computador é entendido como uma máquina que obedece ordens e o programa como uma prescrição de solução para o problema. O conceito central para representação da solução do problema é o conceito de "variável" como uma abstração para uma posição de memória na máquina, para a qual se pode atribuir um valor. O fluxo de controle da execução pela máquina é ditado por sequenciação e por comandos de repetição. Assim, para representar a solução do problema do cálculo do fatorial de um número, o usuário precisa prescrever a solução do problema segundo modificações que dinamicamente alteram os conteúdos das variáveis e conduzem ao resultado do cálculo. Para ilustrar o paradigma procedural a "resolução" de fatorial será escrita em Pascal na sua forma iterativa pois historicamente a iteração ilustra melhor o paradigma da programação procedural. A seguir esse paradigma será retomado para comparação com os demais.

```
Procedure Fatorial(n:integer,var fat:integer);
var i:integer;
begin
        fat:=1;
        for i:=1 to n do fat:=fat*i
end;
```

Esse tipo de representação da solução do cálculo do fatorial envolve uma abordagem ao problema com a máquina em mente. Ou seja, o cálculo do fatorial "acontece" distribuído por ações da máquina que manipulam as variáveis *n* (o objeto do cálculo do fatorial), *i* (variável que auxilia no controle do número de iterações da ação de multiplicar) e *fat* (variável que acumula o resultado da multiplicação). Assim, o cálculo do fatorial de *n* é representado pela ação repetida do comando de atribuição *fat:=fat*i*, cuja semântica explica como é feita a alteração na variável *fat*.
A nível do usuário, a representação da solução do problema nesse meio envolve, além do conhecimento da sintaxe da linguagem, conhecimento da semântica da linguagem ao nível de comando (que modificações nas variáveis um determinado comando provoca) e ao nível de bloco (qual é o "significado" de um conjunto de comandos em relação ao problema em questão).
Portanto, no meio procedural a máquina é tratada como um dispositivo que "obedece" ordens. Assim sendo, a maioria dos comandos é usada para descrever para a máquina "como" resolver o problema. No nosso exemplo, resolver o "problema" Fatorial envolve descrever para a máquina,

em detalhes segundo as restrições da sua linguagem, todos os passos necessários para "cálculo" do fatorial de um número.

O meio funcional

Uma função matemática é um mapeamento de membros de um conjunto (domínio) em outro conjunto (contra-domínio). Ou seja, definir uma função envolve especificar, explícita ou implicitamente seu domínio, seu contra-domínio e o mapeamento que "leva" elementos do domínio a elementos do contra-domínio.

A principal característica do meio funcional é "imitar" o comportamento de funções. Assim, no meio funcional, o computador atua como uma máquina que avalia funções e o programa consiste da definicão e composição de funções. Nas linguagens procedurais, uma expressão é avaliada e seu resultado, em geral, é armazenado em uma célula de memória representada por uma variável. Uma linguagem de programação puramente funcional não usa variáveis ou comando de atribuição. A ordem de avaliação de suas expressões de mapeamento é controlada por recursão e expressões condicionais, enquanto que no meio procedural esse controle é feito por sequenciação e iteração. Dessa maneira, o usuário é "poupado" de preocupações com entidades que o levariam a uma metodologia de mais "baixo nível" (próxima da arquitetura da máquina), para desenvolvimento de programas.

A solução do problema do cálculo do fatorial de um número, por exemplo, é representado no "meio" funcional, por uma função recursiva nos moldes de sua definição matemática, como ilustrado a seguir, em Lisp:

```
(defun fatorial (n)
        (cond
                ((zerop n) 1)
                (T (* n (fatorial (- n 1))))
        )
)
```

A idéia básica não é mais a da repetição de sequência de ações, mas de aninhamento de ativações diferentes da mesma função, cada umas delas "retornando" um valor, que é usado pelas demais, em cadeia, como ilustrado na sequência de expressões abaixo:

```
(fatorial 3)=
(3*(fatorial 2))=
(3* (2* (fatorial 1)))=
(3* (2* (1* (fatorial 0))))=
(3* (2* (1* 1)))=
(3* (2* 1))=
(3* 2)=
(6)
```

Portanto, escrever um "programa", segundo esse paradigma, envolve definir funções e aplicação de funções para o problema em questão e o processo de "execução" pela máquina consiste em avaliar as aplicações das funções envolvidas. O computador assume o papel de uma "máquima funcional".

A nível do usuário, o meio para representação da solução de problemas não é mais baseado nos conceitos de variável, atribuição de valor e iteração; mas, é constituído pelos conceitos de função, comandos condicionais e recursão.

O meio orientado a objetos

O meio orientado a objetos pretende imitar o "mundo real", através do papel do computador como uma máquina que simula a interação entre objetos. Nesse mundo, o programa é constituído dos objetos, mensagens e métodos (possíveis mensagens para as quais um objeto pode responder).

No "meio" orientado a objetos, as unidades de programa são objetos. Por exemplo, desde uma constante numérica até um sistema para manipular arquivos, são todos objetos. Mensagens possibilitam a comunicação entre os objetos e é através delas que uma operação de um objeto é requisitada. Um método especifica a reação de um objeto a uma determinada mensagem recebida correspondente aquele método. Sebesta exemplifica esses conceitos mostrando o significado da expressão *21 + 2* , no paradigma orientado a objetos:

" *o objeto receptor é o número 21, para o qual é enviada a mensagem "+ 2". Essa mensagem passa o objeto 2 para o método "+" do objeto 21. O código desse método usa o objeto 2 para construir um novo objeto, o 23*" (Sebesta, 1988, p. 465).

No paradigma orientado a objetos, a representação do cálculo do fatorial de um número corresponde a um "método" para objetos "inteiros". Esse método, pode ser invocado por uma "mensagem" do tipo *5 fatorial*, construindo o "objeto" *120*. Para ilustrar, mostramos a seguir, a "resolução" de fatorial escrita em Smalltalk (extraída de Sebesta, 1988, p. 473):

```
fatorial
   self = 0
         ifTrue: [^1].
   self < 0
         ifTrue: [self error 'Fatorial não definido']
         ifFalse: [^ self * (self - 1) fatorial]
```

Apesar da "aparência" convencional do código, sua semântica é bem diferente da semântica das linguagens imperativas. As unidades de programa são os objetos, que são uma abstração de dados e possuem a habilidade de herdar características de objetos de classes ancestrais e de se comunicar com outros objetos enviando e recebendo mensagens. O foco da atenção no meio orientado a objetos está colocado, portanto nos objetos e na capacidade de processamento deles, enquanto que na abordagem de programação tradicional o enfoque está nos processos e na implementação deles.

O meio da lógica

Uma das características principais das linguagens para programação em lógica é sua semântica declarativa. A idéia por trás dessa semântica é que existe uma maneira de determinar o significado de cada declaração que não depende de como a declaração seria usada para resolver o

problema. Isto é, o significado de uma dada proposição num programa é determinado a partir da própria proposição, enquanto que, em linguagens imperativas a semântica de um comando requer informações que não estão contidas ou não estão explicitadas no comando. Por exemplo, o conhecimento das regras de escopo para as variáveis é necessário para se entender o significado de determinado comando em um programa escrito numa linguagem procedural.

Assim, seriam declarações válidas no problema do cálculo do fatorial:

"o fatorial de 0 é 1".
"o fatorial de 1 é 1".
"o fatorial de 2 é 2".

etc..

No meio gerado pela programação em lógica, um programa não contém instruções explícitas à máquina. Em vez disso, ele estabelece "fatos" e "regras" sobre a área do problema como um conjunto de axiomas lógicos, que são "interpretados" como "programas". Ilustrando, um "programa" em Prolog, para cálculo do fatorial de um número inteiro pode ser definido por:

```
fatorial(0,1).
fatorial(N,Fat):- N>0,
                  N1 is N-1,
                  fatorial(N1,Fat1),
                  Fat is Fat1 * N
```

No exemplo dado, as duas proposições podem ser lidas como:

"O fatorial de 0 é 1" (e isso é um fato)
"O fatorial de N é Fat **se** o fatorial de N-1 é Fat1 **e** Fat é Fat1 * N" (e isso é uma regra)

Dessa maneira, os programas não estabelecem exatamente "como" um resultado deve ser computado, mas, descrevem fatos e regras que podem levar a máquina à dedução do cálculo do fatorial. O computador assume, portanto, o papel de uma "máquina de inferência", buscando uma prova construtiva para uma meta (pergunta) colocada pelo usuário. A nível do usuário, o computador pode ser visto como uma "lógica" que ele tem acesso para verificação da consistência de suas declarações.

OS PARADIGMAS DESPERTADOS EM LOGO

Logo é uma linguagem de programação proposta por Seymour Papert, cujas raízes derivam de Lisp. A grande contribuição de Papert, pelo que a linguagem é mais conhecida atualmente, está na manipulação de um objeto gráfico, chamado "tartaruga", que é capaz de andar pela tela deixando seu rastro.

Ensinar a tartaruga a fazer algo (por exemplo a figura de um quadrado ou uma casinha) é uma metáfora para a atividade de programar, no contexto da tartaruga. Dessa forma o computador é

abstraído na figura da tartaruga. O resultado (rastro da tartaruga), mostrado na tela fornece um *feedback* para a criança que pode levá-la a reformular o procedimento "ensinado". A beleza dessa idéia está no fato de que no processo de "ensinar" a tartaruga a criança pode refletir o seu próprio processo de aprender e tomar consciência disso faz dela, num certo sentido, uma "epistemóloga"(Papert, 1980).

Programar a tartaruga de Logo é, portanto, um modelo "procedural" de programação, onde o procedimento que a criança cria para "ensinar" a tartaruga deve conter todos os passos que a tartaruga deve executar para conseguir o resultado desejado. O modelo de como a máquina (computador) "funciona" está sendo representado no papel da tartaruga, no sentido de que executa ações sequencialmente. Portanto representar um problema para ser resolvido nesse contexto envolve saber "o quê" a tartaruga é capaz de fazer (primitivas); o que ela "deve" fazer para produzir uma figura na tela e "como" instruí-la a fazer (passo a passo, usando comando repetitivo, etc.). A tartaruga (e indiretamente o computador) é tratada como uma "entidade" que obedece ordens. Esse é, claramente, o paradigma "procedural" de programação.

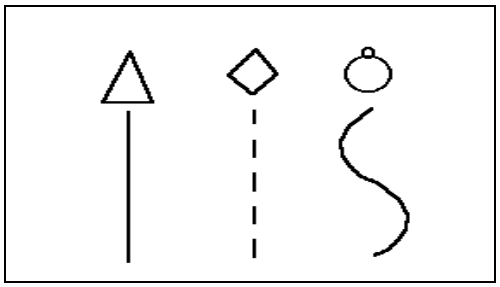
Existe ainda um segundo paradigma do qual Logo se aproximou quando possibilitou o trabalho com múltiplas tartarugas: o paradigma orientado a objetos. Atualmente já existem versoes de Logo orientado a objetos[1] que integra mais essa característica à programação Logo tradicional.

Muir (1989) exemplifica, no contexto de Logo, o paradigma orientado a objetos, através da imagem de um computador contendo várias tartarugas (objetos), cada uma com um formato diferente: uma representada por um pequeno triângulo, outra por um pequeno losango, etc. Cada tartaruga se comporta diferentemente, mesmo que sejam dados a elas comandos idênticos. Cada uma tem seus próprios procedimentos, variáveis, etc. e cada uma pode desenhar com cores diferentes ou padrões diferentes, dependendo de como cada objeto foi definido. Assim, uma delas pode "responder" a um comando "PARAFRENTE" com uma linha contínua de pontos, outra pode responder com uma linha composta de pequenos traços, enquanto que outra "responde" com uma sequência de "ondas", como ilustrado a seguir:

objetos:

mensagem: PARAFRENTE

reação à mensagem:



[1]Uma implementação de Object Logo foi feita pela Coral Software Inc. para Macintosh. Gary Dresher desenvolveu um sistema Logo orientado a objetos, no MIT Lab. de Inteligência Artificial. No Logo para MSX o "sprite" pode ser usado no "meio" orientado a objetos.

Dessa maneira, programação orientada a objetos fornece um ambiente onde, em um único computador, múltiplos computadores podem coexistir. Apesar do novo modelo envolvido no processo de desenvolvimento de programas nessas linguagens,[2] em particular Logo orientado a objetos têm sido usado para introduzir o modelo procedural de programação. A manipulação de várias tartarugas tem sido mais explorada pelo seu efeito estético do que para o trabalho em problemas onde seria interessante um "meio orientado a objetos".

A manipulação de "listas", que é muito pouco compreendida pela comunidade de usuários de Logo, é uma herança da linguagem Lisp[3]. Assim como Logo manipula a "tartaruga", manipula também listas. A diferença é que processamento de listas introduz um novo paradigma de programação: o funcional.

"Lista" é uma estrutura de dados que permite representar uma série ordenada de itens, que podem, por sua vez, ser listas. As primitivas para "manipulação" de listas são, na verdade, funções. Por exemplo:

O primeiro elemento de uma lista é obtido pela aplicação da função (primitiva) PRIMEIRO a uma dada lista - por exemplo, [isto é uma lista]. A palavra "isto" é o resultado da aplicação da função "PRIMEIRO" ao argumento [isto é uma lista].

A nível de unidade os procedimentos definidos pelo usuário podem, por sua vez, ser "funções", pois é permitido que eles "retornem" valores, que podem ser usados por outros procedimentos. Um exemplo de uma função para "retornar" o último elemento de uma lista, em Logo:

```
ap ultimo :lista
se "evazia sp :lista [envie pri :lista]
envie ultimo sp :lista
fim
```

Essa função pode ser "composta" com outras (primitivas ou definidas pelo usuário) como por exemplo:

```
"ji ultimo [a,b,c] [d]"
```

retornando [c,d] como resultado.

Dessa maneira, Logo passa a ter três universos bastante distintos a nível de paradigma de programação: o procedural através da tartaruga (Logo Geométrico), o orientado a objetos através da manipulação de várias tartarugas (Logo Objeto) e o funcional através de listas (Logo Listas).

A PROBLEMÁTICA DA "MISTURA" DE PARADIGMAS EM LOGO

Apesar de considerada "procedural" (Johanson, 1988; Mendelsohn, 1990), em Logo podemos fazer uso, também, do paradigma funcional (definindo funções para manipulação de listas) e do paradigma orientado a objetos (através da manipulação de várias tartarugas). Isso faz de Logo

[2]Outras linguagens de programação com extensões orientadas a objetos existem: Lisp, C, Pascal.

[3]A primeira versão de Logo era conhecida como "baby Lisp" e não tinha a tartaruga e suas primitivas gráficas.

uma ferramenta extremamente flexível e amistosa ao usuário que deseja "sentir o sabor" dos vários paradigmas. Por outro lado, essa multiplicidade de paradigmas, em geral, leva o novato a dificuldades do tipo a usar mecanismos procedurais para "pensar" em funções recursivas, por exemplo. A mudança de um "meio" para outro não parece ser trivial.

A grande dificuldade de usuários Logo "não-experts", no trabalho com listas, a meu ver, está nessa mistura de paradigmas que Logo proporciona. Enquanto o trabalho está centrado no Logo geométrico, o modelo procedural está à tona. A medida em que o novato passa a trabalhar com listas o modelo passa a ser outro e raramente ele toma consciência disso. A dificuldade, frequentemente vem do fato de usar o "meio" procedural para resolver um problema de natureza funcional.

Para ilustrar esse aspecto, apresentamos a seguir duas definições comuns entre usuários não sofisticados, para o problema de "inverter uma lista":

```
ap inverte1 :l
se évazia :l [pare]
esc último :l
inverte1 semúltimo :l
fim

ap inverte2 :l
se évazia :l [pare]
coloque ji pri :l :l2 "l2
inverte2 sempri :l
fim
```

Na primeira solução (**inverte1**) o usuário interpreta "inverter a lista" como uma prescrição à máquina de ações de "escrever os elementos da lista" na ordem contrária. Na segunda solução apresentada (**inverte2**), o aspecto central é a "variável" l2 que deve ser inicializada com uma lista vazia e é dinamicamente alterada pelo comando de atribuição que modifica o conteúdo da variável l2.

Essas definições são típicas de novatos Logo, acostumados ao "meio" procedural da tartaruga. Os procedimentos não são tratados como "funções" que retornam valores. As definições mostradas ilustram a abordagem procedural ao problema e aparentemente o "resolvem". As grandes dificuldades acontecem quando essa abordagem é usada em problemas mais complexos onde o usuário perde o "controle" das modificações que ocorrem dinâmicamente nas variáveis criadas.

Essa dificuldade em "perceber" o "meio" funcional quando trabalhando com listas é menos aparente entre sujeitos com formação em matemática. É comum observar-se em cursos de formação para professores (em geral de 1º e 2º graus) uma maior facilidade de professores de matemática com problemas envolvendo processamento de listas, do que os das demais áreas. Anderson, num artigo em que investiga as dificuldades de estudantes no trabalho com Lisp, relata que curiosamente observou um estudante que, ao contrário dos demais, não teve dificuldade alguma com o aprendizado de programação recursiva: *"Significativamente, era um estudante graduado em matemática que tinha feito uma grande quantidade de trabalho em teoria de funções recursivas"* (Anderson, 1988, p. 162).

Logo tem sido menos discutida como linguagem de programação, e mais discutida como um "ambiente" para uso de computadores em Educação, baseado no trinômio criança-computador(Logo)-facilitador. Nesse "ambiente", Logo é uma máquina virtual que "conversa" com a criança através de seu objeto "tartaruga". O facilitador é a pessoa que "acompanha" a criança em seu trabalho com a máquina. Seu papel no contexto geral do aprendizado da criança é tão difícil quanto é seu grau de interferência nesse processo. Por outro lado, as "idéias poderosas" atribuídas a Logo como recursão, estruturação de procedimentos e *debugging* não são inerentes a Logo (Pea, 1986); elas devem ser entendidas no contexto mais geral dos paradigmas de programação. Assim como o papel do facilitador é de extrema importância no processo de levar a criança à "descoberta" dessas idéias, também o deve ser para levá-la a "perceber" os diferentes paradigmas, adequando a metodologia ao "meio" identificado.

O PARADIGMA DA PROGRAMAÇÃO EM LÓGICA E PROLOG

Segundo o modelo de programar proposto por Prolog, o significado de um "programa" não é mais dado por uma sucessão de operações elementares que o computador supostamente realiza, mas por uma base de conhecimento a respeito de certo domínio e por perguntas feitas a essa base de conhecimento, independentemente. Dessa maneira, Prolog pode ser visto como um formalismo para representar conhecimento a respeito do problema que se quer resolver, de forma declarativa (descritiva). Existe, por trás do programa uma máquina de inferência, em princípio "escondida" do programador, responsável por "encontrar soluções" para o problema descrito.

Para exemplificar essa idéia, imaginemos que o problema em questão seja o de obter todas as combinações possíveis em um cardápio de restaurante, composto de uma entrada, um prato principal e uma sobremesa. Pode-se simplesmente "descrever" os tipos de composição de pratos e a máquina é que se encarregará de "encontrar" as respostas para o que poderia ser uma refeição completa. Assim pode-se definir a seguinte "base de dados":

```
é-refeição(X,Y,Z) se é-entrada(X) e é-prato-principal(Y) e    é-sobremesa(Z).
é-prato-principal(X) se é-carne(X).
é-carne(picanha).
é-carne(galinha).
...
é-entrada(sopa).
é-entrada(frios).
...
é-sobremesa(fruta).
é-sobremesa(doce).
...
```

Para essa base de dados, à "pergunta" é-refeição(X,Y,Z) Prolog responderá com todas as combinações possíveis de entradas, pratos-principais e sobremesas especificados na base de dados. Note-se que o usuário não tem que ocupar-se, em princípio, em "instruir" a máquina, através de comandos apropriados, no sentido de "ensiná-la" a fazer todas as combinações. O foco de sua atenção fica colocado no problema, através da especificação do relacionamento entre os objetos que existem no domínio de seu problema.

Nos paradigmas citados neste texto, as linguagens de programação requerem em maior ou menor grau, que o usuário não-expert faça concessões à máquina de modo que resolver um problema envolve uma abordagem ao problema com a máquina em mente. Com isso a atenção do usuário

é desviada para detalhes de sintaxe e estrutura dos elementos da linguagem, conforme discutido por Johanson (1988):

"*instrução em programação pode ser caracterizada como um pensamento de alta ordem, mas grande parte da energia de aprender a programar é gasto em aprender a sintaxe e estrutura das linguagens de programação e isso deve interferir na meta inicial de desenvolvimento de habilidades em resolução de problemas*" (Johanson, 1988, p.24).

Usuários iniciantes em Prolog tornam-se envolvidos com a especificação de objetos e seus relacionamentos, de forma que a ênfase é colocada na especificação e na análise lógica do conhecimento envolvido no problema; Prolog como "linguagem" praticamente "desaparece". Se por um lado "esconder" do usuário o "raciocínio" que Prolog usa para responder-lhe perguntas (máquina de inferência) traz-lhe uma simplificação que o coloca diretamente envolvido com aspectos relativos ao domínio de conhecimento do problema sendo resolvido, por outro lado, essa prática tem sido uma das maiores responsáveis por levar novatos a concepções errôneas (Mendelsohn, 1990).

A PROBLEMÁTICA DA DUPLA SEMÂNTICA DE PROLOG

Prolog é uma implementação do modelo de computação de programação em lógica em uma máquina sequencial. Assim, o ideal da programação em lógica é somente parte da "história" Prolog.

Várias análises empíricas de tipos de problemas que novatos têm com Prolog têm sido realizadas (Scherz, 1990). Em minha visão muitos dos *bugs* que são apresentados podem ser explicados pelo não entendimento do "meio" gerado pela linguagem, como uma combinação de seus aspectos declarativos e operacioanais.

Um exemplo de uma interpretação errônea da linguagem pelo exagero de seu aspecto declarativo é ilustrado a seguir num protocolo de um novato tentando definir o cálculo do fatorial de um número:

```
fatorial(0,1).
fatorial(N-1,W).
fatorial(N,Z):- Z is N*W.
```

Este protocolo mostra a representação da solução para o problema como uma consequência das declarações:

"o fatorial de 0 é 1.
o fatorial de N-1 é W.
o fatorial de N é Z, que é o produto de N por W".

Parece que nenhum aspecto operacional (como a máquina irá usar essas declarações para responder a pergunta) é suposto.

Frequentemente o novato tem problemas com definições que têm o mesmo significado declarativo e significados operacionais diferentes, como ilustra a base de dados a seguir:

```
idade(tom, 18)
idade(jose,17).
idade(susi,15).
mais_velho1(X,Y):-IX>IY, idade(X,IX), idade(Y,IY).
mais_velho2(X,Y):- idade(X,IX), idade(Y,IY), IX>IY.
```

As duas definições apresentadas (**mais_velho1** e **mais_velho2**), apesar de terem o mesmo significado declarativo, tem significados operacionais diferentes e produzem resultados diferentes quando executadas.

Mostrar o processo de inferência de Prolog gradualmente torna-se necessário a medida em que esse conhecimento amplifica o entendimento do novato sobre o interlocutor no processo.

Por outro lado, usuários alfabetizados em linguagens procedurais, vêem Prolog apenas segundo sua semântica operacional e muito esforço é requerido deles no sentido de "abstrair" a máquina para representar um determinado problema segundo as restrições do próprio problema. Usuários Pascal, por exemplo, tendem a representar problemas em Prolog com a máquina de inferência em mente e têm dificuldades em expressar o problema numa forma declarativa. Entre esses usuários é comum encontrar construções onde o sujeito usa elementos do meio procedural com a sintaxe da nova linguagem, misturando estruturas dos dois paradigmas. Um exemplo que ilustra esse aspecto é apresentado a seguir, na definição da relação "último elemento de uma lista":

último([X],Y):-Y=X ("o último elemento de uma lista com um elemento X é Y se Y é igual a esse elemento X")

O usuário empresta a regra "se-então" do meio procedural em vez de escrever a sentença na sua forma declarativa:

último([X],X) ("o último elemento de uma lista com um elemento X é ele próprio").

Prolog possui uma semântica declarativa, expressa pelas proposições que compõem a sua base de dados, e uma semântica operacional, relativa ao comportamento da máquina de inferência ao buscar uma prova construtiva para uma determinada meta (pergunta). A apropriação do paradigma da programação em lógica, a meu ver, depende da combinação desses dois fatores: o declarativo e o operacional (procedural[4]). Captar o paradigma da programação em lógica e ultrapassar a "porta de entrada" de Prolog envolve, a nível de metodologia, definir ambientes e ferramentas que possibilitem ao usuário trabalhar em uma combinação de ambos os aspectos: o declarativo e o operacional.

DISCUSSÃO E CONCLUSÃO

[4]É importante notar que o termo "procedural" usado neste contexto tem significado diferente do usado para qualificar paradigma procedural de programação. Aqui a máquina em questão é a máquina de inferência de Prolog, e não se refere à arquitetura Von Neuman subjacente.

Na análise feita neste artigo a atividade de "programação de computadores" é considerada uma atividade de "resolução de problemas", onde as linguagens representam, através de seus diferentes paradigmas, os "meios" onde os problemas devem ser resolvidos. Assim, resolver um problema nos paradigmas citados envolve "moldar" o problema segundo as "entidades" representativas de cada paradigma:

- "comandos" que são executados passo a passo pela máquina virtual, se o paradigma for o procedural.

- "funções" , que são aplicadas a certos argumentos e "retornam" valores se o paradigma for o funcional.

- "objetos" que se comunicam se o paradigma for orientado a objetos.

- "proposições" assumidas verdadeiras sobre determinado domínio de conhecimento, se o paradigma for o da programação em lógica.

Se para o profissional do domínio da computação essas mudanças de um meio para outro acontecem de maneira natural, o mesmo não pode ser dito a respeito do novato. A interpretação do novato a respeito do "meio" subjacente à linguagem, que ele usa para representar a solução de um problema, tem implicações no seu processo de resolução do problema e de aprendizado da linguagem. Em geral, o novato num determinado paradigma de programação com experiência anterior no paradigma procedural, usa a sintaxe da nova linguagem sobre uma representação baseada nos elementos do paradigma antigo, no processo de apropriar-se da nova linguagem.

Logo e Prolog têm sido referências obrigatórias quando se fala de linguagens de programação em contextos educacionais. Entretanto, em ambos os casos, a maioria de seus usuários têm grande dificuldade ao ultrapassar o que podemos chamar de suas "portas de entrada": as fronteiras do procedural, incorporadas pela tartaruga, no caso de Logo (Rocha, 1991) e a criação de bases de conhecimento em assuntos de conteúdo puramente declarativo, no caso de Prolog. A grande dificuldade de usuários novatos no trabalho com o paradigma funcional (processamento de listas em Logo, por exemplo) pode ser relativo ao uso do "meio" procedural (de Logo geométrico) para resolver problemas de natureza funcional (Baranauskas, 1990; Baranauskas, 1991; Rocha, 1991). Efeito semelhante acontece aos usuários acostumados a linguagens procedurais quando são introduzidos a outros paradigmas como o caso de Prolog, por exemplo (Baranauskas, 1993).

A apropriação dessas linguagens como meios de expressão pode ser facilitada na medida em que as metodologias e ambientes de desenvolvimento de programas refletirem e explicitarem seus respectivos paradigmas.

Novas linguagens de programação surgirão a partir de novas arquiteturas de computador e de novas abstrações. O paradigma de programação subjacente deveria ser o ponto de partida no *design* de metodologias e ambientes que possibilitem ao usuário a construção conceitual do meio gerado pela linguagem.

REFERÊNCIAS BIBLIOGRÁFICAS

Anderson, J. R.; Pirolli, P; e Farrell, R. (1988) Learning to Program Recursive Functions. Em M.T.H. Chi, R. Glaser, M.J. Farr (Eds.), *The Nature of Expertise.* Hillsdale, N.J.: Lawrence Erlbaum Associates Inc..

Baranauskas, M.C.C. (1990) Procedimento, Função, Objeto ou Lógica? *Relatório Tecnico IMECC*, n. 20/91.

Baranauskas, M.C.C. (1991) Procedure, Function, Object or Logic? Proceedings of the Eighth International Conference o Technology ad Education, Toronto, Otario, pp. 730-731.

Baranauskas, M.C.C. (1993) Criação de Ferramentas para o Ambiente Prolog e o Acesso de Novatos ao Paradigma da Programação em Lógica. Tese de Doutorado, FEE UNICAMP, fevereiro/ 1993, 405p.

Bobrow, D. (1985) If Prolog is the answer, what is the question? or what it takes to support AI programming Paradigms. *IEEE Transactions on Software Engineering*, 11(11)pp1401-1408.

Ennals, R. (1983) *Beginning Micro-Prolog* . Harper and Row,New York.

Johanson. R. P. (1988) Computers, Cognition and Curriculum: Retrospect and Prospect. *J. Educational Computing Research*, 4(1),1-30.

Mendelsohn, P; Green,TRG; e Brna,P. (1990) Programming Languages in Education. *Document TECFA 90-8,* Université de Geneve.

Muir, R, J. (1989) Object-Oriented Programming: Initial Experiences. *The Logo Exchange***.** april 1989.

Papert, S. (1991). New Images of Programming: in search of an educationally powerful concept of technological fluency. (A proposal to the National Science Foundation). Cambridge, MA:MIT Technology Laboratory.

Patterson; J. H. e Smith,M.S. (1986) The Role of Computers in Higher-order Thinking. Em J.A. Culbertson e L.L. Cunningham (ed.), *Microcomputers and Education***,** Chicago: University of Chicago Press.

Pea, R. (1986). Language-independent conceptual "bugs" in novice programming. Journal of Educational Computing Research, 2(1), 25-36.

Rocha, H. V. (1991). Representações Computacionnais Auxiliares ao entedimento de Conceitos de Programação. Tese de Doutorado, FEE UNICAMP, outubro/1991, 500p.

Scherz, Z., Maler, e O., Shapiro, E. (1988). Learning with Prolog - A New Approach. Em J. Nichol, J. Briggs e J. Dean (Eds), *Prolog, Children and Student*, London: Kogan Page.

Sebesta, R. W. (1989) *Concepts of Programing Languages*. California: The Benjamin Cummings Publishing Company, Inc..