Melhorando o Entendimento de Programação usando Esquemas Conceituais em Cursos Introdutórios

Thais Helena Chaves de Castro[1], Crediné Silva de Menezes[2], Alberto Nogueira de Castro Junior[1], Rosane Santos Caruso de Oliveira[2], Maria Cláudia Silva Boeres[2]

[1]Departamento de Ciência da Computação – Universidade Federal do Amazonas (UFAM)
Av. Gal. Rodrigo O. J. Ramos 3000 – 69.077-900 – Manaus – AM – Brasil

[2]Departamento de Informática – Universidade Federal do Espírito Santo (UFES)
Av. Fernando Ferrari, s/n – 29060-970 – Vitória – ES – Brasil

{credine,rcaruso,boeres}@inf.ufes.br

{thais, albertoc}@dcc.fua.br

Abstract. *This paper discusses an experience with programming courses using identification and formal representation of programming schemes and their potential for automatic analysis. A relationship with Bloom's taxonomy has been used to support the classification of these schemes. We intend to use these tools for classification as well as feedback routing with respect to source code produced by programming students.*

Resumo. *Este artigo trata da identificação e representação formais de esquemas de programas e seu potencial para a análise e classificação automáticas. É realizada uma correlação com a taxonomia de Bloom para apoiar a classificação desses esquemas. Planeja-se usar estas ferramentas na construção de ambientes que auxiliem o professor na elaboração de orientações com respeito ao código-fonte produzido por estudantes de cursos introdutórios de programação.*

Palavras-chave*: Ensino de programação; padrões de programa; representação do conhecimento.*

1. Introdução

O aprendizado de programação de computadores constitui-se em uma das mais elaboradas tarefas cognitivas. Ao estudante do assunto é requerida uma visão precisa do problema endereçado, conhecimento de um formalismo apropriado e a habilidade de combinar os elementos adequados para construir uma solução formal. Entre os vários elementos envolvidos neste processo, destacam-se a construção de abstrações e o uso de linguagens artificiais. Estes aspectos vêm, ao longo do tempo, estabelecendo-se como um grande desafio aos professores de disciplinas de ensino introdutório de programação [Mckeown et al, 1999; Joosten et al, 1993; Giegerich et al,1999].

Já se tem observado que no aprendizado inicial de atividades produtoras de artefatos, um fator de grande valia é a possibilidade de o estudante receber críticas, comentários e recomendações sobre as soluções por ele apresentadas (*feedback*). Em

programação, essa meta é atingida parcialmente através do uso de linguagens de programação para as quais exista um ambiente de compilação/interpretação. O analisador sintático produz as críticas com respeito aos erros sintáticos e pode até sinalizar algumas inadequações, como por exemplo, a falta de inicialização de variáveis. Por outro lado, ao submeter um programa à execução o estudante pode comparar os resultados obtidos com os esperados e assim avaliar a correção do seu programa. Alguns ambientes podem ainda fazer essas comparações a partir de um plano de teste. Enfim, é possível e já se tem usado o apoio das máquinas na produção de *feedback*.

Contudo, o tipo de resposta que o computador fornece, embora útil, representa apenas uma pequena fração das críticas e orientações necessárias para que se atinja o domínio completo da arte de programar computadores. Em primeiro lugar, o interpretador não confere os resultados produzidos por um programa com os resultados esperados, ou seja, nada diz sobre a correção do programa. Em segundo lugar, nada é dito sobre a qualidade da solução obtida. Se para o primeiro tópico, já se pode contar com ferramentas verificadoras de resultados, é importante enfatizar que o segundo constitui-se em um grande desafio para os professores e monitores de cursos introdutórios. Sabe-se que quantidade e variedade de soluções que em geral precisam ser analisadas dependem da capacidade de gerá-las por parte dos alunos e da natureza da "matéria-prima" utilizada. A análise e a avaliação das soluções apresentadas são de grande importância no processo de amadurecimento do estudante, contribuindo para que o mesmo se acostume com o questionamento sobre o produto obtido, e que por certo, carece de um modelo adequado.

Normalmente, cursos introdutórios são caracterizados pela ausência de análise adequada das soluções apresentadas - um verdadeiro perigo à formação de um profissional competente. Em geral, o estudante fica muito satisfeito com o fato de seu programa funcionar. O ideal é que a solução do estudante seja criticada incisivamente, mostrando de forma clara o tipo de inadequação cometida quanto à qualidade do programa apresentado. Conseqüentemente, a tarefa de avaliação torna-se bastante árdua, já que, para o desenvolvimento satisfatório do estudante é preciso que ele resolva uma quantidade significativa de problemas de programação.

O trabalho aqui relatado é parte de um esforço multi-institucional que busca definir caminhos para atenuar a tarefa de avaliação. Buscamos os elementos necessários para a construção de assistentes capazes de analisar as soluções apresentadas, classificá-las e a elas responder com um comentário adequado à compreensão do estudante, conforme apresentado em [Castro et al, 2002]. A partir da uma análise de uma coleção de problemas resolvidos por alunos foi feita uma classificação das variações de solução, o que nos permitiu a elaboração de uma estrutura conceitual. Na seqüência formalizamos este conhecimento com o objetivo de ter uma descrição do conhecimento a ser usado pelos agentes de software que se pretende construir. Foram utilizados no presente trabalho, programas descritos em programação funcional, produzidos por estudantes em seu primeiro curso de programação. A linguagem de programação utilizada é Haskell [Bird & Wadler, 1988], uma linguagem funcional.

2. Identificando Esquemas de Programa

Para identificar as classes de soluções apresentadas pelos estudantes, definiu-se como etapa inicial desse trabalho, a análise das soluções apresentadas nas resoluções de suas

listas de exercícios das disciplinas introdutórias de programação funcional, dos cursos de Engenharia da Computação e Ciência da Computação, de duas Instituições de Ensino Superior brasileiras ao longo de três semestres letivos, totalizando seis turmas de calouros de uma instituição e duas da outra.

Quando do início dos trabalhos, observou-se que não havia necessidade de se verificar todas as listas de exercícios (o que seria um esforço muito grande, sem maiores ganhos), pois seria perfeitamente possível identificar padrões de programas, inicialmente, apenas com alguns exercícios. Face os objetivos de nosso trabalho, apenas as soluções que produzem resultados corretos para o plano de teste foram consideradas. Para cada exercício selecionado, os professores responsáveis construíram uma solução, aqui denominada de solução base, tendo em conta o nível de maturidade dos estudantes e os princípios que haviam sido apresentados e discutidos nas aulas. Procedeu-se então o estudo e, para cada exercício selecionado foi feito um levantamento dos vários tipos de solução.

Do conjunto de soluções disponíveis foi realizada uma triagem, para eliminar as soluções redundantes, produzindo-se com isso, uma lista de soluções distintas. Tomando-se como referência a solução base, cada solução proposta pelos estudantes foi analisada, visando encontrar elementos que possibilitem estabelecer diferenças entre elas, permitindo que, com isso, fosse construída uma árvore das variações encontradas para cada exercício. Em alguns casos, a solução de um estudante possibilitou a identificação de uma nova possibilidade de resolução de problemas (ou uma variação mais elegante), e quando isto ocorreu, as soluções iniciais foram redefinidas e as soluções daquele item do trabalho, reclassificadas.

2.1. Subconjunto de Haskell considerado

Para este estudo consideramos apenas os temas abordados na fase inicial do curso, cobrindo o trecho da linguagem apresentado a seguir através de sua gramática.

<definição> ::= <nome> {parâmetro} = <expressão> [<definição local>]
<expressão> ::= <expressão simples> | <expressão condicional>
<expressão condicional> ::= **if** <expressão predicativa> **then** <expressão> **else** <expressão>
<definição local> ::= **where** <definição>

3. Esquemas Conceituais – natureza do conhecimento

Como resultado das sucessivas etapas de classificação e organização das soluções apresentadas pelos estudantes, foi possível classificar as variações de soluções em ***variações sintáticas*** e ***variações cognitivas***. As variações sintáticas referem-se às diferenças entre as formas de construção obtidas a partir das possibilidades permitidas pela linguagem. Quanto às cognitivas, pode-se dizer que estão relacionadas com os diferentes modelos mentais que os estudantes constroem para elaborar a solução de um mesmo problema. Foram percebidas também variações semânticas, que não são analisadas neste trabalho por não serem o foco principal desta pesquisa. A seguir, apresenta-se uma discussão mais detalhada dos dois primeiros tipos de variação citados

e uma sugestão de uso da figura de ***evidenciação*** como estratégia para obtenção de variação semântica.

3.1. Variações Sintáticas

É importante que se identifique às variações sintáticas de uma solução para que possamos identificá-las como soluções equivalentes à solução base. As principais variações sintáticas identificadas foram:

i. Uso desnecessário do <if...then...else>, quando a expressão era do tipo boolean. Por exemplo:

   f x = if x>0 then True else False quando bastaria usar f x = x>0;

ii. Uso desnecessário de parêntesis. Por exemplo:

   if x>0 then (if ...) else ... onde bastaria usar if x>0 then if ... else ...;

iii. Uso diferenciado do <if...then...else> no que diz respeito ao domínio de interpretação do problema.

   Exemplo 1: if x>0 then 1 else 0 ou if x<0 then 0 else 1.

   Exemplo 2: if x>0 then <exp1> else <exp2> ou if not x>0 then <exp2> else <exp1>;

iv. Uso da comutatividade dos operadores. Exemplo: <exp1>&&<exp2> é similar a <exp2>&&<exp1>

3.2. Variações Cognitivas

Entende-se por variações cognitivas aquelas que dizem respeito diretamente à forma de estruturação da solução pelo estudante. Neste ponto do aprendizado, espera-se que os conceitos presentes na solução estejam diretamente relacionados com os conceitos encontrados na descrição do problema ou que possam ser diretamente depreendidos desta.

De um ponto de vista mais geral, podemos identificar a existência de diferentes metáforas para um dado problema. Cada metáfora é determinada por um conjunto de conceitos. Estes conceitos podem ser registrados pelo estudante, em diferentes níveis de abstração, produzindo, com isso, uma variedade de soluções. Entendemos que a percepção de abstrações no grau adequado produzirá soluções de melhor qualidade. Portanto, é de extrema relevância que o estudante seja incentivado e recomendado a usá-las.

A fim de fornecer um aporte teórico para a pesquisa relatada neste trabalho, baseamo-nos na taxonomia de Bloom [Bloom, 1956], no que concerne a níveis de competências cognitivas (níveis de abstração) relacionados a algumas demonstrações de habilidades típicas para cada nível. Essas competências são:

a. conhecimento
b. compreensão
c. aplicação
d. análise

e. síntese

f. avaliação

Ao se estudar pela primeira vez sobre um tópico de história, por exemplo, onde apenas se consegue visualizar datas, eventos e lugares, vislumbrando uma visão bem abrangente da situação estudada, está-se na fase mais básica do desenvolvimento cognitivo (a). Já quando se vai aprofundando mais nas leituras, começa-se obter o entendimento das questões relativas ao tema, bem como seu significado, podendo até predizer conseqüências (b). Com o decorrer de outras leituras, já se consegue utilizar as informações obtidas e aplicá-las em um outro contexto. Nesta fase, atingiu-se a etapa (c). As duas etapas seguintes ocorrem quase que simultaneamente. Na primeira (d), reconhece-se os significados escondidos ("lê-se as entrelinhas"), identificando-se novos componentes; na segunda (e) generaliza-se a partir de fatos conhecidos e já se pode tirar conclusões a respeito do tema. Na última fase do desenvolvimento cognitivo (f) o sujeito compara e discerne as idéias, faz escolhas baseadas em argumentos raciocinados, reconhece a subjetividade dos fatos e consegue valorar as evidências.

Neste trabalho, partindo da classificação das habilidades cognitivas citadas anteriormente, identificamos alguns de percepção de um determinado conceito, evidenciados em programas de alunos dos cursos sujeitos do experimento:

i. ausência de registro

ii. destaque

iii. nomeação

iv. parametrização

v. generalização

A situação em que o estudante não deixa qualquer vestígio de que tenha se apercebido do conceito é chamado de ausência de registro (i), relacionado à competência (*a*). Já o primeiro indício de que a abstração foi levada em conta é quando o estudante explicita através do uso de mecanismos de parentetização (ii), relacionado à competência (*b*). Ao dar um nome, o estudante deixa claro, principalmente pela escolha deste nome, que o conceito abstrato foi capturado (iii), relacionado à competência (*c*), já que essa ação propicia a oportunidade de utilização de um mesmo conceito mais de uma vez. Com a parametrização (iv) atinge-se o nível mais elevado da percepção de conceitos e de suas interações, relacionado às competências (*d* e *e*). Com a criação de funções paramétricas auxiliares o mesmo conceito pode ser utilizado com mais flexibilidade em diferentes situações. A generalização (v) pode trazer benefícios para a clareza do programa, mas não se constitui, necessariamente, em um artifício que o torna melhor que os demais, por produzir quando mal utilizada, códigos confusos. Esta última é, por isso, uma característica relacionada à competência (*f*).

Algumas vezes, a solução apresentada pelo estudante não deixa clara como os principais elementos relevantes à solução foram capturados, com respeito ao domínio do problema. Outras vezes identifica uma total despreocupação com a eficiência e a clareza da solução. Estas situações podem ser mais bem entendidas através da seqüência de soluções para um determinado problema, mostrada na Tabela 1.

Observando as soluções da Tabela 1, nota-se que as soluções 3 e 4 ressaltam o fato de que *a* e *b* possuem a mesma propriedade. Isto não é mostrado com clareza nas soluções 0, 1 e 2. A solução 0 apresenta uma solução simplista do problema, não capturando abstrações relevantes do domínio do mesmo. Essa solução não exibe de forma clara a idéia de abstração de intervalo. Como a solução desejada deve guardar informações sobre o problema, e tendo em vista que isto não ocorre na solução 0, ela é considerada como inflexível. A solução 1 apenas destaca a abstração de intervalo, o que mostra indícios de que o estudante percebeu a relevância do conceito. Em 2 observa-se não apenas o destaque da abstração, como também sua nomeação. Em 3 identifica-se uma parametrização (tal solução facilita sua reutilização sempre que necessário). E, finalmente, em 4 encontra-se uma generalização do problema apresentado incluindo os registros de abstração identificados nas soluções anteriores. A solução 4 inclui ainda o tratamento geral de intervalos através da função descritora de intervalos (*pertence*).

3.2.1. Generalização ×Reutilização

A solução 4 da Tabela 1, descrita na subseção anterior, oferece algumas vantagens sobre as demais. Estas vantagens podem ser consideradas tanto do ponto de vista de entendimento do leitor (legibilidade), quanto do ponto de vista de modificação do problema (reutilização). Ilustraremos esta situação através de propostas de modificação no problema original. Para cada uma apresentamos a nova solução, evidenciando a naturalidade com que o *script* é ajustado para a nova requisição. Entende-se por script um conjunto de definições de funções necessária à resolução de um determinado problema.

i. Transformação do problema 1 para um outro intervalo fechado (Tabela 2).

Enunciado: Avalie se, dados dois números *a* e *b*, ambos estão no intervalo [2..8].

Tabela 1 - Variações Cognitivas de um Problema

Problema 1 – Avalie se, dados dois números, *a* e *b*, ambos estão no intervalo [0..6]	
Solução 0	*Ausência de Registros*
Prob1 a b = a>=0 && a<=6 && b>=0 && b<=6	
Solução 1	*Destaque*
Prob1 a b = (a>=0 && a<=6) && (b>=0 && b<=6)	
Solução 2	*Nomeação*
prob a b = prob1 && prob2 where prob1 = a>=0 && a<=6 prob2 = b>=0 && b<=6	
Solução 3	*Parametrização*
prob a b = prob1 a && prob1 b where prob1 x = x>=0 && x<=6	

Solução 4	*Generalização*
pertence x ni nf = x>=ni && x<=nf -- prob a b = prob1 a && prob1 b where prob1 x = pertence x 0 6	

Tabela 2 – Mudança de Valores

Solução 4ª
pertence x ni nf = x>=ni && x<=nf -- prob a b = prob1 a && prob1 b where prob1 x = pertence x 2 8

ii. Transformação do problema 1 para um um intervalo aberto (Tabela 3).

Enunciado: Avalie se, dados dois números, *a* e *b*, ambos estão no intervalo (0..6).

Tabela 3 - Intervalo Aberto

Solução 4b
pertence x ni nf = x>ni && x<nf -- prob a b = prob1 a && prob1 b where prob1 x = pertence x 0 6

iii. Transformação do problema 1 para parametrizar a função com três elementos (Tabela 4).

Enunciado: Avalie se, dados três números *a*, *b* e *c*, os três estão no intervalo [0..6].

Tabela 4 - Função com Três Elementos

Solução 4c
pertence x ni nf = x>=ni && x<=nf -- prob a b c = prob1 a && prob1 b && prob1 c where prob1 x = pertence x 0 6

3.2.2. Generalizações Inconvenientes

Embora a generalização seja interessante para o caso apresentado na seção anterior, nem sempre seu uso pode ser considerado apropriado. O caso a seguir ilustra uma dessas situações.

Problema 2: Dados 3 números, *a*, *b* e *c*, determine a média aritmética dos extremos.

É fácil perceber que o problema apresenta, em seu enunciado, a citação de duas constantes: (i) *a quantidade de números a serem considerados (n)* e, (ii) a *quantidade de números sobre os quais determinaremos a média aritmética*.

Neste caso, podemos destacar os seguintes aspectos:

- O valor de *n* interfere na quantidade de construções do tipo *if-then-else* a serem usadas;
- Nesse estágio, outros recursos da linguagem não são dominados pelo estudante, o que inviabiliza a construção de soluções mais genéricas;
- Conseqüentemente, a generalização é cabível com relação ao domínio do problema, mas não se encaixa no domínio da solução pela falta de recursos;
- É preciso, neste caso, restringir-se aos dados do problema.

Para o domínio do problema aqui apresentado, destacam-se dois conceitos relevantes que podem ser facilmente capturados na descrição da solução: o menor número e o maior número. Há ainda, obviamente, a abstração *média aritmética*, conforme é mostrado na Tabela 5. É comum também uma solução um pouco menos refinada, com o uso da parentetização, conforme ilustra a Tabela 6.

Tabela 5 - Abstração de "menor" e "maior"

Solução 2ª
mediaDosExtremos a b c = mediaArit where mediaArit = (menor + maior) / 2.0 menor = if a<b then if a<c then a else c else if b<c then b else c maior = if a>b then if a>c then a else c else if b>c then b else c

Tabela 6 - Parentetização

Solução 2b
mediaDosExt a b c = ((if a<b then if a<c then a else c else if b<c then b else c) + (if a>b then if a>c then a else c else if b>c then b else c)) / 2.0

3.3. Evidenciação – uma estratégia para melhorar a legibilidade

Intuitivamente, a idéia de evidenciação de uma solução aparece como um tipo de variação cognitiva relevante, o que foi constatado analisando os programas dos estudantes. Antes de discutirmos alguns exemplos, é interessante afirmar o que

consideramos uma evidenciação. É o isolamento de um termo que aparece várias vezes em uma descrição, como na expressão aritmética abaixo:

$$a * b + (c + d) * b + h + (e + f - 3) * b + g$$

onde a multiplicação por *b* aparece várias vezes. Sua evidenciação nos levaria a:

$$h + g + [a + (c + d) + (e + f - 3)] * b$$

Vejamos ainda um outro exemplo, agora com um programa em Haskell, mostrado na Tabela 7.

Tabela 7 - Metade do Menor Número

Problema 3: Dados dois números, a e b, determine a metade do menor deles.	
Solução 0	*Comentário*
prob3 a b = if a<b then a / 2.0 else b / 2.0	O estudante perspicaz notará rapidamente que esta solução pode ser reescrita, evidenciando a divisão por 2.0, produzindo a solução 1.
Solução 1	*Comentário*
prob3 a b = (if a<b then a else b) / 2.0	Esta operação destaca a abstração "menor de dois números", através da expressão (if a<b then a else b). Se ainda, essa operação é nomeada, chega-se à solução seguinte.
Solução 2	*Comentário*
prob3 a b = menor / 2.0 where menor = if a<b then a else b	Se quisermos refiná-la ainda mais podemos generalizá-la, como mostra a solução 3.
Solução 3	*Comentário*
menor x y = if x<y then x else y -- prob3 a b = menor a b / 2.0	Com esta solução, foi atingido um alto grau de refinamento.

Como podemos observar, partimos da intenção em isolar a divisão por 2.0 e acabamos por atingir uma generalização, através da valorização do conceito *menor*, que anteriormente não era explicitado. A conclusão que chegamos neste caso é que o tipo de solução que atingimos inevitavelmente também se manifestaria se tivéssemos buscado a identificação de conceitos relevantes no domínio do problema. Por outro lado, acreditamos também que quando produzimos uma solução, podemos usar a operação de evidenciação como uma fonte de busca de conceitos ocultos. Sendo assim, decidimos considerá-la como uma heurística na árdua tarefa de construir soluções abstratamente convenientes.

Percebemos ainda que há fortes indícios de que a evidenciação seja uma ferramenta de grande utilidade quando estivermos trabalhando no domínio de soluções, buscando aquelas que possuam um melhor desempenho computacional. Por exemplo, no caso da expressão que apresentamos no início desta discussão, observa-se uma redução de duas operações de multiplicação.

Ilustremos um pouco mais a situação, revisitando o problema 2, mostrado no Quadro 1.

Quadro 1 - Revisitando o Problema 2

Problema 2 – Dados 3 números, *a*, *b* e *c*, determinar a média aritmética dos extremos

Solução 2a

```
mediae a b c = if (a>=b) && (a>=c)
                then caso1
                else if (b>=a)&&(b>=c)
                        then caso2
                        else caso 3
                 where
                        caso1 = (a + (if b>=c then c else b)) / 2.0
                        caso2 = (b + (if a>=c then c else a)) / 2.0
                        caso3 = (c + (if a>=b then b else a)) / 2.0
```

Solução 2b

```
mediae a b c = (if (a>=b)&&(a>=c)
                  then caso1
                  else if(b>=a)&&(b>=c)
                          then caso2
                          else caso3) / 2.0
                where
                        caso1 = (a + (if b>=c then c else b))
                        caso2 = (b + (if a>=c then c else a))
                        caso3 = (c + (if a>=b then b else a))
```

4. Formalização dos Esquemas Conceituais (modelagem do conhecimento)

Para a representação do conhecimento será utilizada a lógica de primeira ordem, por ser uma linguagem capaz de expressar univocamente os pressupostos sugeridos pela pesquisa.

4.1. Estruturação

Cada variante metafórica de um problema possui uma lista de conceitos, os quais temos expectativas que o estudante abstraia. Para descrever uma solução, usa-se *scripts*, os que podem ser modelados pela utilização de componentes sintáticos. Cada solução possui um identificador. Escolhemos modelar a descrição de uma solução utilizando dois símbolos relacionais: *CompCognitivo* e *CompSintatico*. Ambos binários, representando a associação entre uma solução e um componente cognitivo ou sintático, respectivamente. Cada componente sintático e cada componente cognitivo serão descritos por uma relação associando um identificador com uma descrição. A primeira denominamos de *elemSintatico* e a segunda de *elemCognitivo*. As diversas variações, sintática ou cognitiva, são modeladas pelos símbolos predicativos *TipoVarSintatica* e *TipoVarCognitiva*, respectivamente.

CompCognitivo(<solução>,<componente cognitivo>)
CompSintatico(<solução>,<componente sintático>)
ElemSintatico(<identificador>,<descrição>)
ElemCognitivo(<identificador>,<descrição>)
TipoVarSintatica(<tipo de variação sintática>)
TipoVarCognitiva(<tipo de variação cognitiva>)

Solução(<solução>)

As soluções estão ligadas entre si por uma ou mais variações sintáticas ou por uma ou mais variações cognitivas. No primeiro caso modelamos usando o símbolo predicativo *VarSintatica* e no segundo *VarCognitiva*. Em qualquer uma das duas relações envolvemos as duas soluções, o componente e o tipo de variação.

VarSintatica(<solução 1>, <solução 2>, <componente sintático>, <tipo de variação>)
VarCognitiva(<solução 1>, <solução 2>, <componente cognitivo>, <tipo de variação>)

4.2. Axiomatização

Nesta etapa do trabalho identificamos alguns elementos para a descrição mais apurada do conhecimento envolvido na descrição de soluções e suas interações. A seguir, listamos o resultado de uma primeira exploração das interrelações e suas particularidades. Entendemos que há muito ainda para ser identificado, o que esperamos atingir em projetos futuros.

1) a relação de variação sintática é irreflexiva;
2) a relação de variação cognitiva é irreflexiva;
3) a relação de variação sintática é anti-simétrica;
4) a relação de variação cognitiva é anti-simétrica;
5) Cada solução não pode possuir mais de uma origem de derivação sintática;
6) Cada solução não pode possuir mais de uma origem de derivação cognitiva;
7) Entre duas soluções, as variações cognitivas em um dado conceito, são únicas;
8) Entre duas soluções, as variações sintáticas em um dado componente, são únicas;
9) Toda solução, exceto a solução abstrata (solução 0), possui uma solução da qual deriva sintática ou semanticamente;
10) As soluções abstratas só possuem variantes cognitivas metafóricas;
11) As soluções abstratas não possuem variantes sintáticas.

4.2.1. Representação Lógica dos Axiomas

ax1: $\forall$s1, s2, c1, t1 VarSintatica(s1,s2,c1,t1) $\Rightarrow$ $\neq$(s1,s2)
ax2: $\forall$s1, s2, c1, t1 VarCognitiva(s1,s2,c1,t1) $\Rightarrow$ $\neq$(s1,s2)
ax3: $\forall$s1, s2, c1, t1 VarSintatica(s1,s2,c1,t1) $\Rightarrow$ $\sim\exists$c2,t2 VarSintatica(s2,s1,c2,t2)
ax4: $\forall$s1, s2, c1, t1 VarCognitiva(s1,s2,c1,t1) $\Rightarrow$ $\sim\exists$c2,t2 VarCognitiva(s2,s1,c2,t2)
ax5: $\forall$s1, s2, s3, c1, c2, t1, t2 VarSintatica(s1,s2,c1,t1) $\wedge$ VarSintatica(s3,s2,c2,t2) $\Rightarrow$ =(s1,s3)
ax6: $\forall$s1, s2, s3, c1, c2, t1, t2 VarCognitiva(s1,s2,c1,t1) $\wedge$ VarCognitiva(s3,s2,c2,t2) $\Rightarrow$ =(s1,s3)
ax7: $\forall$s1, s2, s3, c1, c2, t1, t2 VarSintatica(s1,s2,c1,t1) $\wedge$ VarSintatica(s3,s2,c2,t2) $\Rightarrow$ =(c1,c2) $\wedge$ =(t1,t2)
ax8: $\forall$s1, s2, s3, c1, c2, t1, t2 VarCognitiva(s1,s2,c1,t1) $\wedge$ VarCognitiva(s3,s2,c2,t2) $\Rightarrow$ =(c1,c2) $\wedge$ =(t1,t2)
ax9: $\forall$s solução(s) $\wedge$ $\neq$(s,solução-0) $\Rightarrow\exists$s1,c1,t1 VarSintatica(s1,s,c1,t1)
$\vee$ $\exists$s2,c2,t2 VarCognitiva(s2,s,c2,t2)
ax10: $\forall$s,s1,c,t solução(s) $\wedge$ =(s,solução-0) $\wedge$ VarCognitiva(s,s1,c,t)
$\Rightarrow$ =(t,metafórica)
ax11: $\forall$s,s1,c,t solução(s) $\wedge$ =(s,solução-0) $\Rightarrow$ $\neg\exists$s1, c, t VarSintatica(s,s1,c,t)

4.2.2. Aplicação dos Axiomas

Os axiomas apresentados na seção anterior foram definidos após a observação do desempenho de quatro turmas de alunos calouros, totalizando dois semestres letivos. Após esses resultados iniciais, aplicamos o conhecimento inferido às outras turmas de que trata esta pesquisa, incrementando os procedimentos experimentais com testes de lógica, questionários e entrevistas.

Os resultados obtidos até o presente momento confirmam a axiomatização descrita e confirmam a validade dessa classificação para os outros conceitos abordados nesses cursos introdutórios, como o uso de listas e os paradigmas aplicativo e recursivo.

5. Conclusões

Neste trabalho buscamos identificar componentes importantes das soluções apresentadas por estudantes de disciplinas introdutórias de programação, dentro de uma abordagem funcional, visando compará-las com soluções consideradas mais adequadas do ponto de vista da legibilidade.

A identificação e a classificação das modalidades de variações cognitivas possibilitaram a modelagem das interligações entre soluções, definindo uma linguagem de primeira ordem. Ademais, vislumbra-se uma metodologia para o ensino de programação, voltada para o grau de legibilidade e reutilização de uma solução. Nesta direção, podemos mostrar ao aluno como comparar soluções visando melhorar as qualidades acima citadas.

Além da extensão dos axiomas apresentados na Seção 4.2 e a preparação de material para cursos de programação apoiados nas idéias relatadas, a construção de ferramentas baseadas em conhecimento modelado conforme ilustrado aqui, é um desdobramento natural desse trabalho.

Referências

Bird, R.S., Wadler, Ph., Introduction to Functional Programming, Prentice Hall, ISBN 0-13-484197-2, 1988, New York.

Bloom, B. S. (ed.) *Taxonomy of Educational Objectives: the classification of educational goals. Handbook I – cognitive domain*. 1956, New York.

Castro, T., Castro Jr, A. N., Menezes, C. S., Boeres, M. C. S., Rauber, M. C. P. V. Utilizando Programação Funcional em Disciplinas Introdutórias de Computação In: XXII Congresso da Sociedade Brasileira de Computação / X Workshop sobre Educação em Computação, 2002, Florianópolis.

Giegerich, R., Hinze R., Kurtz, S. Straight to the Heart of Computer Science via Functional Programming. Workshop on Functional and Declarative Programming in Education. 1999. Paris, FR.

Joosten, S., Van-den-Berg, K., Van-der-Hoeven, G. Teaching Functional Programming to First-Year Students. Journal of Functional Programming. Vol.3, N.1. 1993.

Mckeown, J., Farrell, T. Why We Need to Develop Success in Introductory Programming Courses. CCSC - Central Plains Conference. 1999. Maryville, MO.