Representando Padrões em um Ambiente de Apoio à Aprendizagem de Programação

Thais Helena Chaves de Castro [1], Alberto Nogueira de Castro Júnior [2], Crediné Silva de Menezes [1]

[1]Mestrado em Informática – Universidade Federal do Espírito Santo (UFES)

Caixa Postal 01-9011 – 29060-970 – Vitória – ES – Brasil

{thaish, credine}@inf.ufes.br

[2]Departamento de Ciência da Computação – Universidade Federal do Amazonas (UFAM)

Av. Gal. Rodrigo O. J. Ramos, 3000 – 69077-900 – Manaus – AM – Brasil

albertoc@dcc.fua.br

Resumo. Este artigo descreve uma pesquisa sobre padrões para representação de programas no contexto do aprendizado de programação por estudantes iniciantes. Ao longo de uma experiência de cerca de dez anos de ensino de programação para iniciantes, colecionamos classes de problemas e esqueletos de programas desenvolvidos pelos estudantes. Portanto, no contexto da arquitetura SAAP, descrevemos uma proposta de linguagem ou pseudo-linguagem para a representação dessas classes de problemas e esquemas de programas utilizando a linguagem funcional Haskell.

Palavras Chave: padrões de programa, esquemas de programa, programação funcional.

1. Introdução

No aprendizado de programação está envolvido o desenvolvimento de diferentes habilidades, principalmente a capacidade de abstração. Para desenvolver essa habilidade optamos pela utilização de uma linguagem funcional, que julgamos mais adequada a esse primeiro contato do estudante com programação, estimulando-o a escrever programas para resolver problemas geométricos.

Após anos aplicando disciplinas introdutórias, vimos que a utilização de esqueletos ou esquemas de programas pode auxiliar professores e estudantes no melhor entendimento dos conceitos de programação. Na próxima seção (Seção 2) mostramos uma visão geral da pesquisa em padrões e esquemas de programa. Na Seção 3 damos direções de como representar padrões em linguagem funcional. Finalmente, na Seção 4 propomos uma linguagem para representação de esquemas de programa, utilizando Haskell.

2. Padrões e Esquemas de Programa

Em diversos trabalhos são descritas representações de programas ou trechos de programas através de esquemas genéricos. Nesta seção, mostramos uma visão geral sobre o uso de esquemas de programa, relacionando-os aos trabalhos mais relevantes para o aprendizado de programação.

Uma forma clássica de derivação de esquemas é a semi-formal, como a apresentada em [Dershowitz 1985], onde são mostrados passos para abstrair um esquema a partir de um ou mais programas, por analogias, onde as invariantes e provas de correção desempenham um papel importante. O primeiro a

fazer é tentar identificar exatamente quais ocorrências de símbolos nos programas são análogas. Então, as transformações devem ser feitas somente para as mais relevantes. O problema é que para isto ser possível, precisa-se de informação sobre como os programas originais foram construídos.

O processo de abstração começa com um conjunto de programas, que, segundo o autor, pode ser um ou vários. É aceitável a abstração a partir de um único programa, de alguma forma escolhendo símbolos de especificação para serem abstraídos e permanecendo para que sejam examinadas as conseqüências. O uso de mais programas reduz a arbitrariedade, garantindo que somente partes análogas sejam abstraídas.

Já em [Michaelson 1996] são mostrados padrões informais em linguagem funcional (ML), descrevendo meta operações, como operações com listas. O trabalho tem o objetivo de facilitar o aprendizado dos estudantes iniciantes em programação, tentando eliminar as mal-compreensões.

Seguindo a mesma linha de trabalho que Michaelson, em [Bieliková e Návrat 1998] é argumentado que o estudante aprende mais com uma abordagem baseada em esquemas, através da instanciação de modelos gerais, apresentados previamente a ele. Após um dado esquema de programa ser introduzido, é solicitado que os estudantes instanciem o esquema como um exercício de escrita e, depois, como um exercício de programação.

Esse trabalho de Bieliková e Návrat apresenta esquemas de programa conhecidos em LISP e Prolog, sem formalismos ou regras de derivação, mostrando apenas algumas etapas para sua elaboração, refinando as generalizações.

Em outro trabalho interessante [Abd-El-Hafiz 1997] é apresentado um estudo comparativo entre duas técnicas de decomposição de programas. A primeira é a decomposição de *loops* em eventos, que são fragmentos parcialmente ordenados. A outra é *program slicing*, um método de decomposição automática de um programa, analisando seu controle e fluxo de dados. As três principais diferenças entre *slices* e eventos são quanto à representação, ordem e tamanho, mas uma vantagem em um compensa uma desvantagem em outro.

Como descrito por [Bieliková e Návrat 1998], esquemas informais para iniciantes precisam se restringir a problemas nos quais: (i) os únicos tipos de dados são listas, incluindo átomos (números ou símbolos); (ii) sem recursão indireta; (iii) sem operações com efeito colateral (como entrada/saída).

Apesar de haver essas restrições de abstração, que precisam ser graduais, para representações informais de esquema, entendemos que esse tipo de representação é mais adequado ao processo ensino-aprendizagem de programação, pela possibilidade de se utilizar a própria linguagem de programação adotada em uma disciplina para descrever tais esquemas.

3. Representando Padrões

Como visto na seção anterior, esquemas têm sido largamente utilizados para representar padrões de programas em linguagens imperativas e em lógica. Porém, nesta pesquisa, não encontramos padrões de programas agrupados de acordo com níveis de complexidade, relacionados à progressão natural na aprendizagem de estudantes em cursos introdutórios de programação.

Nas instituições onde atuamos utilizamos o paradigma funcional na primeira disciplina de programação (Haskell), que é ministrada para três cursos: Ciência da Computação, Engenharia da Computação e Matemática, conforme descrito em [Castro et al 2002a]. Ao longo dessa experiência de 10 anos, fomos identificando (i) a necessidade de agruparmos os problemas estudados em diferentes classes, representando níveis de experiência dos estudantes no decorrer da disciplina; e (ii) padrões de respostas semelhantes entre si, instanciados a essas classes de problemas.

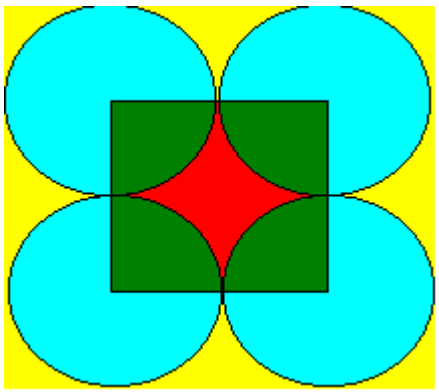
Na etapa inicial, o curso de programação contempla problemas geométricos, por serem de fácil visualização. Estes problemas podem ser agrupados em, pelo menos, duas classes (Quadro 1): (i) problemas de determinação; e (ii) problemas de pertinência. Partindo dessas classes iniciais, abstraímos

alguns esquemas de programas de respostas de estudantes a listas de exercícios, aplicadas nos últimos semestres.

Para descrever os padrões de soluções precisamos de uma linguagem. Um dos requisitos é que esta linguagem seja amistosa o suficiente para que o próprio professor, que já conhece os principais tipos de programas de uma determinada classe de problemas, possa descrever seus esquemas em alto nível. Estes padrões serão então transformados em esqueletos de programa em Haskell, a partir dos quais as soluções dos alunos serão analisadas.

Problema: Considere dado um quadrado, paralelo aos eixos cartesianos, através de um ponto ***P*** onde suas diagonais se interceptam e um lado ***l***. Considere que cada um de seus cantos é o centro de uma circunferência de raio igual à metade do lado do quadrado. Pede-se:

a) uma função para determinar a área do quadrado que circunscreve o conjunto formado pelo quadrado e a s circunferência, que não é coberta pelo conjunto (determinação);
b) dado um ponto P1 qualquer, escreva uma função verifica se P1 pertence à região interior ao quadro e exterior às circunferências (verificação).



Quadro 1 - Exemplo de problemas geométricos com questões de verificação e determinação

4. Proposta de uma Linguagem para Representação de Padrões

Por não existir uma representação de padrões para problemas geométricos nem em Haskell nem em outra linguagem de programação, propomos uma linguagem para descrever as classes desses problemas e, posteriormente, outras classes a serem acrescentadas por um professor.

A linguagem proposta está nos moldes dos esqueletos de programa em Prolog descritos por [Bowles et al 1994], contendo meta-funções, através da formação de *pattern*, no lugar da lógica de segunda ordem utilizada para os programas em Prolog. Essa linguagem foi modelada em BNF, sendo posteriormente submetido ao ALEX , um analisador léxico escrito em Haskell, e está em fase de validação no Happy, um validador de gramáticas.

Os padrões representados nessa linguagem serão incorporados no ambiente SAAP [Castro et al 2002b], constituindo-se em uma ferramenta para autoria de esquemas, contendo uma interface para um professor de uma disciplina introdutória poder representar os padrões encontrados por ele, utilizando uma classe de problemas já representada ou criando uma nova, sem precisar se ater a detalhes específicos da linguagem de programação.

5. Referências

ABD-EL-HAFIZ, S. *Effects of Decomposition Techiques on Knowledge-Based Program Understanding*. In: Proceedings of the International Conference on Software Maintenance. IEEE. 1997.

BIELIKOVÁ, M. e NÁVRAT, P. *A Schema-Based Approach to Teaching Programming in Lisp and Prolog*. In: Proceedings of PEG International Conference. pp. 22-29. 1997.

BIELIKOVÁ, M. e NÁVRAT, P. *Use of Program Schemata in Lisp Programming: an Evaluation of its Impact on Learning*. In: Informática Journal, Vol. 9, Num. 1, pp. 5-20. 1998.

BOWLES, A.; ROBERTSON, D.; VASCONCELOS, W.; VARGAS-VERAS, M. e BENTAL, D. *Applying Prolog programming Techniques*. In: International Journal Human-Computer Studies, Vol 41, pp. 329-350. 1994

CASTRO, T.; CASTRO Jr, A.; MENEZES, C.; BOERES, C. e RAUBER, C. *Utilizando Programação Funcional em Disciplina Introdutórias de Computação*. In: Proceedings of Workshop de Educação em Computação-CSBC. 2002a.

CASTRO, T.; CASTRO Jr, A.; MENEZES, C. e CURY, D. *Arquitetura SAAP – Sistema de Apoio ao Aprendizado de Programação*. In: Proceedings of Workshop de Informática na Escola-CSBC 2002b.

DERSHOWITZ, N. *Program Abstraction and Instantiation*. In: ACM Transactions on Programming Languages and Systems, Vol. 7, Num. 3, pp. 446-477. 1985.

MICHAELSON, G. *Automatic Analysis of Functional Program Style*. In: Proceedings of the Australian Software Engineering Conference. IEEE. 1996.