Um tutor inteligente para o ensino/aprendizado de programação com técnicas de diagnóstico hierárquico baseado em modelos.

Wellington R. Pinheiro[1], **Leliane N. de Barros**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
Rua do Matão, 1010 – Cidade Universitária – CEP 05508-090 – São Paulo – SP

{wrp,leliane}@ime.usp.br

Abstract. *Model based diagnosis from Artificial Intelligence is a technique that has been used to detect faulty components in physical systems as well as faults in software systems. Although this approach is useful to help experienced programmers to detect faults in theirs programs, it must be adapted to be used with novice programmers. In this papers, we show an approach to automatic debugging that explores abstract components (functions, procedures or elementary patterns) to perform an hierarchical program diagnose. This way, the hypotheses of failure are communicated to the student at each abstract level, increasing his opportunity of learning.*

Resumo. *Diagnóstico baseado em modelos é uma técnica de Inteligência Artificial usada para detectar componentes falhos em dispositivos físicos bem como, falhas em sistemas de software. Embora essa abordagem possa auxiliar programadores experientes a encontrar falhas em seus programas, para ser usada por aprendizes de programação ela precisa ser adaptada. Nesse artigo, apresentamos uma abordagem para depuração automática que explora a idéia de componentes abstratos (funções, procedimentos ou padrões elementares) para fazer o diagnóstico de programas de forma hierárquica. Assim, as hipóteses de falha são comunicadas ao aluno em cada nível de abstração, aumentando as suas oportunidades de aprendizado.*

1. Introdução

Sabe-se que aprender como combinar sentenças de uma linguagem de programação para construir um programa é uma tarefa difícil [Winslow 1996]. Isto é, especificar um método de resolução de problemas conhecido, em uma linguagem de programação conhecida, ainda é a principal dificuldade enfrentada por aprendizes de programação. Essa dificuldade pode ser observada nas disciplinas de Introdução à Programação para turmas de exatas na Universidade de São Paulo, nas quais 10% a 40% dos alunos são reprovados. Isso sugere que, para se ensinar os princípios de programação é necessário explorar todas as formas de ferramentas de apoio ao aprendiz.

Um *Sistema Tutor Inteligente* (ITS - *Intelligent Tutoring System*) [Wenger 1987] é uma ferramenta de aprendizado eletrônico que usa técnicas de Inteligência Artificial. Um ITS deve capaz de tomar decisões instrucionais, e.g.: quando um aluno deve aprender mais sobre um mesmo tópico, ou mudar para um novo tópico.

PROUST [Johnson and Soloway 1984] é um ITS de programação que tenta identificar planos gerais de programação como fragmentos do programa do aluno. Esses planos de programação são previamente armazenados em uma biblioteca. Essa abordagem apresenta duas desvantagens: (1) necessidade de um especialista para a especificação dos planos gerais de programação e; (2) não há garantia de que todas as possíveis soluções para um problema serão cobertas pela biblioteca de planos.

Um sistema de depuração automática de programas [Mateis et al. 2000, Mayer et al. 2002] utiliza técnicas de Diagnóstico baseado em Modelos (MBD - *Model Based Diagnosis*) [de Kleer and Williams 1987, Reiter 1987] para encontrar falhas em software: de um lado, está o sistema (composto de componentes) cujo comportamento falho pode ser observado; do outro lado, está um modelo descrevendo o comportamento correto do sistema que é usado para fazer predições a seu respeito. Em [Barros et al. 2005], essa técnica foi adaptada para ser utilizada por aprendizes de programação. Nesse trabalho, um sistema tutor "tenta entender" a diferença entre o programa do aluno e suas intenções (estratégia de solução). O programa é visto como um sistema que deve ser diagnosticado. Observe que o modelo correto do sistema não está disponível: existem diversos programas que podem resolver um único problema. A idéia é usar informações fornecidas pelo aluno, sobre suas intenções, para detectar as partes do programa com erros. A vantagem dessa abordagem em relação à adotada pelo PROUST é que não é necessário usar uma biblioteca de planos e metas previamente construída para um conjunto de problemas.

Um aspecto interessante na depuração automática de programas de alunos é que o processo de depuração, por si próprio, pode promover o aprendizado: um efeito colateral que ocorre nas metodologias de ensino/aprendizado centradas na resolução de problemas. Em [Delgado 2005], foi proposta uma etapa de discriminação de hipóteses de falha em que o aluno interage com o sistema, fornecendo suas predições sobre o comportamento do programa, em termos do estado das variáveis. Apesar dessa técnica de depuração ser bastante promissora para ser usada no apoio ao aprendizado de programação, existem algumas limitações, entre elas:

1. Mesmo para um programa pequeno, o sistema de diagnóstico pode apontar diversas sentenças e expressões do programa que podem estar falhas. Apresentar um conjunto grande de hipóteses de falha ao aluno pode confundi-lo, ao invés de ajudá-lo, na correção do programa.
2. Apenas mostrar linhas do programa que contêm erros, traz pouca informação que possa auxiliar um aprendiz a encontrar os erros. Isso pode dificultar a compreensão das falhas e, conseqüentemente, não promover o aprendizado.
3. Um aluno iniciante nem sempre sabe fornecer informações detalhadas sobre o comportamento de seus programas, i.e. em termos do estado das variáveis.

Nesse trabalho, propomos um método para a depuração de programas mais apropriado para ser usado como ferramenta de aprendizado de programação. Essa proposta estende nosso trabalho anterior sobre diagnóstico de programas com MBD [Barros et al. 2005, Delgado 2005] para considerar a abordagem hierárquica [Mozetič 1991, Chittaro and Ranon 2004]: uma variação do MBD em que o modelo do sistema é descrito por componentes abstratos que podem ser sucessivamente decompostos. Acreditamos que, introduzindo a noção de abstrações e refinamentos hierárquicos, seja possível tratar as limitações citadas anteriormente.

Esse artigo está organizado conforme segue. Na Seção 2, apresentamos os fundamentos sobre diagnóstico baseado em modelos, diagnóstico hierárquico e depuração de programas baseada em modelos. Na Seção 3, apresentamos a nossa proposta para fazer depuração de programas com o diagnóstico hierárquico. Finalmente, na Seção 4, apresentamos as nossas conclusões.

2. Conceitos básicos

2.1. Diagnóstico Baseado em Modelos (MBD)

Diagnosticar é a tarefa de identificar a causa de falhas num sistema físico que se manifesta em um comportamento observado [Benjamins 1993]. Essa tarefa envolve as sub-tarefas: detecção de sintomas, geração de hipóteses e discriminação de hipóteses. A geração de hipóteses é formalizada a seguir [Reiter 1987].

Definição 2.1. *Um sistema é dado por $\langle SD, COMP\rangle$, sendo SD a descrição do sistema e COMP um conjunto finito de constantes representando os componentes do sistema. A descrição do sistema é feita por um conjunto de sentenças da LPO, descrevendo o comportamento dos componentes do sistema (*modelo comportamental*) e as conexões entre os componentes do sistema (*modelo estrutural*).*

Definição 2.2. *Seja OBS um conjunto de observações (valores das conexões do sistema) representadas por sentenças em lógica de primeira ordem (fatos). Um problema de diagnóstico $P_{MBD} = \langle SD, COMP, OBS\rangle$ consiste em encontrar um conjunto $\Delta \subseteq COMP$, que quando considerados falhos, expliquem as observações.*

Definição 2.3. *Um diagnóstico para $\langle SD, COMP, OBS\rangle$, que obedece o princípio de parcimônia, é um conjunto minimal $\Delta \subseteq COMP$, tal que:*

$$SD \cup OBS \cup \{ok(C) | C \in COMP - \Delta\} \cup \{\neg ok(C) | C \in \Delta\}$$

é consistente, sendo $ok(C)$ um fato representando que o componente C tem um comportamento correto (não está falho).

Definição 2.4. *Um conjunto de contribuintes para $\langle SD, COMP, OBS\rangle$ é um conjunto $CO \subseteq COMP$ tal que $SD \cup OBS \cup \{ok(C) | C \in CO\}$ é inconsistente.*

Definição 2.5. *Um* conjunto de corte *de uma família de conjuntos F é um conjunto $H \subseteq \bigcup_{S \in F} S$ tal que $H \cap S \neq \emptyset$ para todo $S \in F$ (i.e., H contém pelo menos um elemento de todo $S \in F$). Um conjunto de corte H é minimal se e somente se não existir $H' \subset H$, $H' \neq H$, que também seja um conjunto de corte.*

Se F é a família de todos os conjuntos de contribuintes para um problema de diagnóstico, então cada conjunto de corte minimal de F é uma possível explicação (diagnóstico) para todas as observações. O Teorema 2.1 mostra uma forma construtiva de encontrar as hipóteses de falha a partir dos conjuntos de contribuintes.

Teorema 2.1. *$\Delta \subseteq COMP$ é um diagnóstico para $\langle SD, COMP, OBS\rangle$ se e somente se Δ é um conjunto de corte minimal para a coleção de conjuntos de contribuintes de $\langle SD, COMP, OBS\rangle$* [Reiter 1987].

Uma breve revisão sobre o algoritmo de Reiter. Reiter [Reiter 1987] propôs um algoritmo de diagnóstico que computa todos os conjuntos de corte minimais para uma família de conjuntos de contribuintes F. O algoritmo gera um grafo acíclico com nós rotulados por conjuntos de F e arcos rotulados por elementos desses conjuntos. A idéia é que para

cada nó rotulado por um conjunto *S*, os arcos saindo desse nó sejam rotulados por elementos de *S*. Seja $H(n)$ o conjunto formado pelos rótulos dos arcos no caminho de um nó *n* até a raiz do grafo. O nó *n* deve ser rotulado por um conjunto *S* tal que $H(n) \cap S = \emptyset$. Se não houver um conjunto que atenda essa restrição, o nó é rotulado por @. Cada caminho terminado por @ corresponde a um conjunto de corte minimal.

2.2. Diagnóstico Hierárquico Baseado em Modelos (HMBD)

Abstrações têm sido usadas para reduzir o custo computacional de se encontrar um diagnóstico em sistemas físicos [Mozetič 1991]. Em geral, as abstrações são usadas na forma de hierarquias de componentes que descrevem um sistema.

Diagnóstico hierárquico baseado em modelos (HMBD) é uma abordagem de MBD que usa múltiplos níveis de detalhamento para descrever um sistema. Cada nível, exceto o primeiro, é composto de abstrações construídas a partir do nível mais detalhado. Dois tipos de abstrações são normalmente usadas [Chittaro and Ranon 2004]: (i) *abstrações comportamentais*, definidas através do agrupamento das descrições comportamentais do sistema, e.g. um circuito digital opera com 0s e 1s mas seus componentes internos operam com valores analógicos; (ii) *abstrações estruturais*, construídas através da agregação de componentes, e.g. um circuito digital pode ser visto como uma agregação de transistores, resistores, etc. Um componente definido por uma abstração estrutural é chamado de *componente abstrato*. Chamamos de *modelo base* um modelo sem componentes abstratos.

Um componente abstrato *C* representa um conjunto de *componentes internos*, $AGR(C)$, sendo sua especificação dada por: (i) *modelo estrutural interno*, que descreve como os componentes internos estão conectados; (ii) *modelo comportamental*, que descreve como o componente abstrato deve se comportar em termos de seus componentes internos; (iii) *modelo estrutural externo*, que descreve as conexões do componente abstrato com os demais componentes do sistema.

A Figura 1, mostra dois modelos de um circuito digital. O modelo na Figura 1(a), é composto por um componente *A*1 que representa uma porta "e" lógica; e um componente inversor *I*1. A Figura 1(b) mostra o componente abstrato *NA*1, que é composto a partir da agregação dos componentes *A*1 e *I*1.

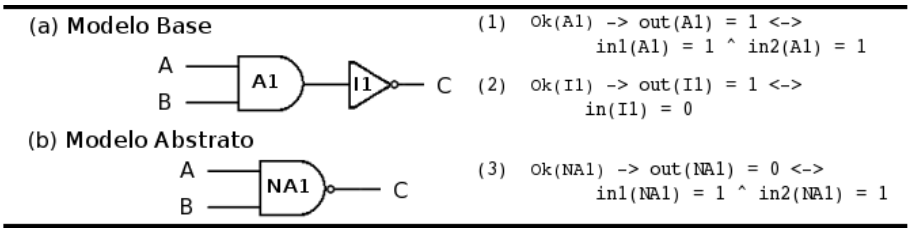

Figura 1. Exemplo de agregação estrutural.

Uma propriedade importante do diagnóstico hierárquico é que se um componente abstrato tem seu comportamento definido como correto, então todos os componentes internos de *CA* também devem estar corretos, i.e.:

$$ok(CA) \quad \rightarrow \bigwedge_{C \in AGR(CA)} ok(C) \qquad (1)$$

Note que esse axioma impede que sejam detectadas falhas quando uma falha em um componente interno que compensa a falha em outro componente interno.

Dada a descrição de um sistema com vários níveis de abstração, o método usado no HMBD para encontrar as hipóteses de falha consiste em mapear cada nível de abstração em um problema de diagnóstico não hierárquico e usar um método tradicional de MBD para encontrar as hipóteses de falha em cada nível [Chittaro and Ranon 2004]. Começando do nível mais abstrato, as hipóteses de falha encontradas são usadas para excluir componentes dos níveis mais detalhados, conforme o Axioma 1. Assim, quando for feito o diagnóstico no nível mais detalhado, um conjunto menor de componentes será considerado na geração de hipóteses.

2.3. Depuração de Programas Baseada em Modelos (MBSD)

A tarefa da depuração de programas é encontrar falhas de execução em programas. Em particular, falhas funcionais são falhas que resultam do armazenamento de um valor incorreto em alguma variável, em algum dos possíveis fluxos de execução do programa. Exemplos de falhas funcionais são: omitir um operador (e.g. escrever i ao invés de $i+1$); uso de um operador incorreto (e.g. i++ ao invés e ++i); ou variáveis incorretas (e.g. $a[i]$ ao invés de $a[j]$); não inicialização de variáveis; condições de seleção erradas. Falhas estruturais, por outro lado, são falhas no código fonte que alteram a estrutura do programa em questão. Por exemplo: a ausência de sentenças, sentenças fora de ordem, sentenças desnecessárias ou o acesso a variáveis incorretas. Nesse trabalho, trataremos falhas funcionais e alguns casos específicos de falhas estruturais.

Para fazer o diagnóstico de programas, ao invés do diagnóstico de sistemas físicos, a descrição do sistema deve ser derivada do programa do aluno e da semântica da linguagem. Esse modelo deve representar componentes (sentenças e expressões), suas conexões (variáveis) e seus comportamentos, baseado no comportamento (incorreto) do programa do aluno. As observações são saídas incorretas para um determinado caso de teste, em diferentes pontos do programa. Existem dois tipos de modelos que podem ser usados para fazer a modelagem de programas: *modelo baseado em valor* [Mateis et al. 2000] e *modelo baseado em dependência* [Mayer et al. 2002]. Nesse trabalho, utilizaremos o modelo baseado em valor.

O modelo baseado em valor. Nesse modelo, expressões e sentenças são representadas como componentes, tendo suas semânticas descritas por conjuntos de sentenças lógicas (modelo comportamental). Componentes são conectados somente se existir um fluxo de informação entre as sentenças ou expressões correspondentes, e.g. existe um fluxo de informação entre uma atribuição A_1 e outra sentença S_1 se a atribuição modifica o valor de uma variável v que é acessada por S_1 e não há outra sentença que modifique o valor de v no fluxo de execução entre A_1 e S_1 [Mateis et al. 2000]. As sentenças lógicas definindo essas conexões correspondem ao modelo do programa do aluno. Assim, o modelo estrutural pode ser construído da seguinte forma: (i) atribuições, condicionais, laços, instruções do tipo `return`, chamadas de funções ou procedimentos, e expressões são todas mapeadas em componentes; (ii) para cada valor atribuído a uma variável é criada uma conexão que representa o valor corrente da variável. Se uma expressão E usa uma variável v, então o componente que representa E deve usar a conexão que representa o valor atual da variável v.

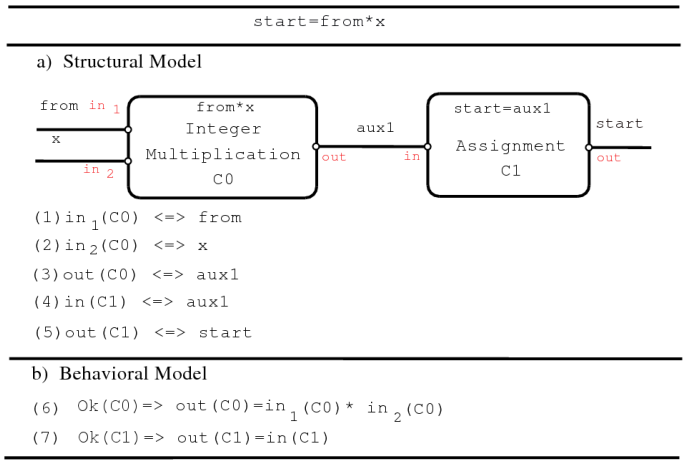

Figura 2. Visão simplificada dos modelos: estrutural e comportamental, de uma sentença de um programa em Java [Barros et al. 2005].

A Figura 2(a) mostra o modelo estrutural baseado em valor para a sentença: `start = from * x`. Um componente de atribuição tem uma porta de entrada e uma porta de saída. A porta de entrada representa o resultado da expressão do lado direito da atribuição, e a porta de saída representa o novo valor da variável no lado esquerdo da atribuição. O operador de multiplicação é mapeado em um componente com duas portas de entradas e uma porta de saída; a porta de saída está relacionada com a avaliação da operação. O modelo comportamental é derivado da semântica da linguagem de programação. A Figura 2(b), mostra o modelo comportamental do componente *IntegerMultiplication* ($C0$) e do componente *Assignment* ($C1$), sendo que, $in_1(C0)$ e $in_2(C0)$ representam as entradas do componente $C0$ e $out(C0)$ é a saída do componente $C0$. O modelo baseado em valor de um programa com falhas pode ser usado pelo algoritmo de Reiter para encontrar as hipóteses de falha.

3. Depuração Hierárquica de Programas

A idéia do uso de HMBD para a depuração de programas é permitir que componentes abstratos sejam usados na comunicação com o aluno. Para isso, é necessário identificar abstrações no programa do aluno, que podem ser: (i) procedimentos ou funções; (ii) *padrões elementares*, que são estratégias para resolver problemas de programação e podem ser usadas para auxiliar alunos novatos no aprendizado de programação; seu uso é recomendado por educadores de programação [Porter and Calder 2003].

O uso de componentes abstratos em um ITS de programação pode trazer as seguintes vantagens: (i) estabelecer o diálogo com o aprendiz em termos de estratégias de resolução de problemas, i.e., através de uma linguagem de comunicação de alto nível; (ii) permitir o raciocínio sobre o programa de uma forma hierárquica,

3.1. Exemplo de um modelo abstrato de programa

Problema da multa. *Dados três inteiros: (1) o valor de uma prestação, (2) um inteiro* 1 *ou* 0 *informando se deve ou não ser aplicada uma multa de atraso e; (3) um inteiro* 1 *ou* 0 *informando se deve ou não ser aplicada a multa de mora. O valor da prestação deve ser calculado da seguinte forma: se multa for igual a 0, decremente 25 da prestação, senão,*

```
      1    public static void main(String[] args) {
      2        int valor, multa, mora, valorTotal;
      3        valor = Integer.parseInt(args[0]);
      4        multa = Integer.parseInt(args[1]);
      5        mora = Integer.parseInt(args[2]);
      6        valorTotal = valor;
+     7        if (multa == 1)
+     8            valorTotal = valorTotal + 50;
+     9        else valorTotal = valorTotal -25;
*    10        if (mora == 1)
*    11            valorTotal = valorTotal - 5;
*    12        else valorTotal = valorTotal + 15;
     13        System.out.println(valorTotal);
     14    }
```

Figura 3. Programa para solucionar o problema da multa, com erro na linha 12.

incremente 50. Além disso, caso o valor da mora seja 0, decremente 5 do valor obtido no cálculo da multa, caso contrário, incremente 15.

A Figura 3 apresenta o programa de um aluno, escrito na linguagem Java, para resolver o *problema da multa*. Note que o aluno usou duas vezes nesse programa um padrão elementar de *Seleção Alternativa* [Bergin 2004] (linhas 7 e 10). Esse padrão elementar pode ser aplicado quando se deseja executar diferentes ações em condições complementares. Uma vez que o padrão elementar foi aplicado, é interessante que o sistema de depuração o utilize como um componente abstrato.

A partir do programa do aluno é gerado o modelo baseado em valor de nível 0 (modelo base) e a partir desse modelo é construído um modelo abstrato. A Figura 4 mostra esses dois modelos. O modelo abstrato da Figura 4(a) possui os componentes abstratos $CA1$ e $CA2$ que representam os padrões elementares identificados no programa. Esses componentes abstratos têm como componentes internos $\{C4, C5\}$ e $\{C10, C11\}$, respectivamente. Para cada componente abstrato é necessário especificar os modelos estruturais interno e externo; e o modelo comportamental. A Tabela 1 descreve os modelos do componente abstrato $CA1$.

3.2. O algoritmo HMBSD

Na abordagem tradicional de MBSD (Seção 2.3), a interação com o usuário ocorre somente durante a discriminação de hipóteses, enquanto que, no HMBSD (*Hierarchical Model based Software Debugging*), a interação pode ocorrer em cada nível de abstração. Assim, as hipóteses de falha são apresentadas ao aluno logo após serem encontradas em cada nível de abstração, e para cada hipótese o aluno poderá:

1. *Reconhecer que aquela hipótese corresponde de fato ao seu erro, modificando em seguida o programa que será novamente verificado com os casos de testes*. Caso o programa modificado ainda apresente falhas, o aluno pode reiniciar o processo

Estrutural externo	**Estrutural interno**	**Comportamental**
$in_1(CA1) = novoValor3$ $in_2(CA1) = multa1$ $in_3(CA1) = const1$ $out_1(CA1) = novoValor5$	$composit(C,E,Cond) \rightarrow$ $in_2(C) = in_1(E) \wedge in_3(C) = in_2(E) \wedge$ $in_1(C) = in_1(Cond) \wedge result(E) = aux4 \wedge$ $out_1(C) = out_1(Cond) \wedge condResult(C) = aux4$ $composit(CA1, C4, C5)$	$cond(C) \wedge ok(C) \rightarrow \exists E, Cond\,[$ $cond - composit(C,E,Cond) \wedge$ $ok(E) \wedge ok(C)\,]$ $cond(CA1)$

Tabela 1. Modelos descrevendo o componente abstrato $CA1$.

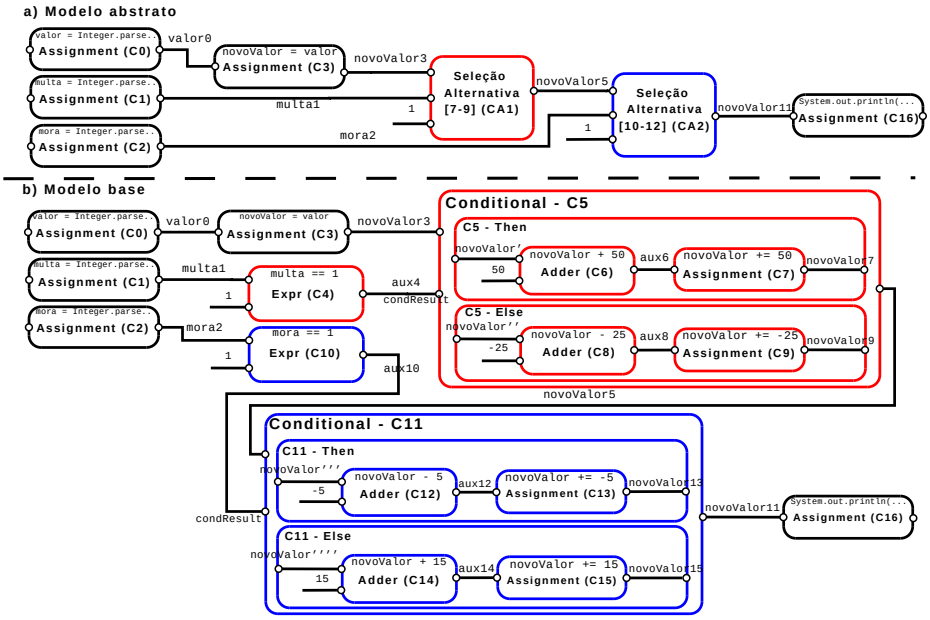

Figura 4. Modelos nos níveis 0 e 1 do programa na Figura 3.

de depuração no programa modificado ou voltar à versão anterior do programa e escolher outra hipótese de falha para continuar com a depuração.

2. *Pedir que o sistema de depuração o ajude a verificar se a hipótese de falha em questão corresponde de fato ao erro do programa*. Nesse caso, o aluno deve fornecer informações adicionais a respeito de suas intenções, e.g.: o valor esperado para uma variável durante a execução do programa. Caso a intenção do aluno corrobore com as predições do sistema de depuração, uma nova hipótese será apresentada ao aluno. Caso contrário, recaímos no caso 1.
3. *Optar por fazer um diagnóstico mais detalhado, ao detectar um componente abstrato com uma possível falha*. Para isso, o aluno pode requerer que o sistema substitua um componente abstrato pelos seus componentes internos (*refinamento*), para que seja feito o diagnóstico em um modelo mais detalhado. Nesse caso, um novo conjunto de hipóteses de falha é gerado.

O Algoritmo 1, descreve essa interação acima para a depuração de programas.

3.3. Exemplo de depuração com o HMBSD

Seja *CT* um caso de teste para o problema da multa com entradas: $valor = 1000$, $multa = 0$ e $mora = 0$; e saída esperada 970. Executando o programa do aluno na Figura 3 com *CT*, obtemos a saída $990 \neq 970$, indicando que há falhas no programa. Após uma análise, verificamos que existem dois tipos possíveis de falhas: (i) a condição na linha 10 deveria ser `valorTotal != 1` (falha funcional); ou (ii) as sentenças nas linhas 11 e 12 estão trocadas (falha estrutural).

Utilizando a técnica MBSD para fazer a depuração do programa, são obtidas a seguintes hipóteses de linhas com falha: $\{\{6\},\{7\},\{9\},\{10\},\{12\}\}$. Note que a linha que contém a falha funcional real do programa (linha 10) está entre as hipóteses de falha. Mostraremos a seguir como obter as hipóteses de falha com o Algoritmo 1, considerando os componentes abstratos identificados no programa.

Algoritmo 1: HMBSD(P, A, T)

input: P: programa do aluno; A: abstrações identificadas em P; T: casos de testes para P.
construir os modelos para o programa P, considerando as abstrações A;
fazer o diagnóstico do programa considerando o modelo mais abstrato (M')
enquanto *houver alguma hipótese de falha no programa, considerando o modelo* M' **faça**
  apresente uma hipótese falha para o aluno e aguarda pela sua decisão;
  se *o aluno optou pela ajuda do sistema para verificar a hipótese atual* **então**
    colete novas observações do aluno;
    se *as novas observações são discrepantes das predições* **então**
      o aluno conseguiu encontrar a(s) falha(s) e deve modificar o programa;
    senão o aluno deve continuar a discriminação de hipóteses com as observações atuais;
  se *o aluno optou por modificar o programa* **então**
    se *o programa modificado não tem falhas* **então** termine o processo de depuração;
    senão se *o aluno quiser tentar corrigir as falhas do programa na versão anterior à modificação* **então**
      volte à versão anterior do programa e continue o processo de depuração;
    senão reinicie o processo de depuração considerando as modificações do aluno no programa;
  senão se *o aluno optou por refinar um componente abstrato CA do conjunto de hipóteses atual* **então**
    substitua em M' os modelos de CA pelos modelos de seus componentes internos;
    considerar M' como esse modelo modificado e continuar o processo de depuração;
fim

O HMBSD constrói os modelos do programa e faz o diagnóstico no modelo mais abstrato (Figura 4(a)). O diagnóstico obtido nesse nível é:

$$\{\{6\},\{\mathit{SelecaoAlternativa}[7-9]\},\{\mathit{SelecaoAlternativa}[10-12]\}\}$$

Além das linhas do programa possivelmente falhas, essas hipóteses de falha também incluem componentes abstratos, e.g., $\mathit{SelecaoAlternativa}[10-12]$ indica que o padrão elementar de Seleção Alternativa, aplicado nas linhas de número 10 a 12, deve estar falho. Essa hipótese de falha é comunicada ao aluno através de um linguagem de alto nível, da seguinte forma: *"A condição do padrão elementar de Seleção Alternativa usado (linhas 10-12) fez com que o ramo else do condicional fosse executado, mas isso gerou um erro na saída do programa"*. Após comunicar a falha ao aluno, suponha que ele decida verificar as falhas do programa em um nível mais detalhado, considerando dessa vez as instruções que compõem o componente abstrato representado por $\mathit{SelecaoAlternativa}[10-12]$. O sistema de depuração substitui os modelos do componente que representa a $\mathit{SelecaoAlternativa}[10-12]$ ($CA2$ no modelo da Figura 4(a)) pelos modelos de seus componentes internos e executa novamente o diagnóstico. Nesse caso, as hipóteses devolvidas pelo sistema de diagnóstico são:

$$\{\{6\},\{\mathit{SelecaoAlternativa}[7-9]\},\{10\},\{12\}\}.$$

Note que, entre esse novo conjunto de hipóteses gerado, está a linha com a falha estrutural mencionada anteriormente. Além disso, para todos os níveis de abstração são geradas menos hipóteses de falha do que na técnica de MBSD tradicional.

4. Conclusão

Nesse artigo, apresentamos um método para a depuração de programas (HMBSD) com o objetivo de superar algumas limitações do método tradicional de depuração de programas baseado em modelos (MBSD). O método proposto usa a técnica de diagnóstico hierárquico baseado em modelos e a representação de componentes abstratos identificados nos programas do aluno, tais como: funções, procedimentos e padrões elementares. Esses componentes abstratos são usados para representar o programa em vários níveis de abstração. Para cada nível de abstração é possível fazer a discriminação de hipóteses considerando os componentes abstratos, o que possibilita que as falhas sejam comunicadas

ao aluno através de uma linguagem de alto nível. Com essa abordagem um aluno poderá reconhecer suas falhas durante o processo de depuração antes de chegar do nível das instruções da linguagem de programação. Atualmente, estamos trabalhando na integração do algoritmo apresentado em uma das ferramentas de aprendizado eletrônico, o Sakai (`http://www.sakaiproject.org/portal`), para que ele possa ser avaliado em termos pedagógicos com alunos de uma turma de Introdução à Programação.

References

Barros, L. N., Mota, A. P. S., Delgado, K. V., and Matsumoto, P. M. (2005). "A tool for programming learning with pedagogical patterns". In *eclipse '05: Proc. of the 2005 OOPSLA workshop on Eclipse technology eXchange*. ACM Press.

Benjamins, R. (1993). *Problem Solving Methods for Diagnosis*. PhD thesis, University of Amsterdam.

Bergin, J. (2004). "Patterns for selection", http://csis.pace.edu/ bergin/patterns /Patternsv4.html. Acessado em: 08/06/2009.

Chittaro, L. and Ranon, R. (2004). "Hierarchical model-based diagnosis based on structural abstraction". *Artificial Intelligence*, 155(1-2):147–182.

de Kleer, J. and Williams, B. C. (1987). "Diagnosing multiple faults". *Artificial Intelligence*, 32(1):97–130.

Delgado, K. V. (2005). Diagnóstico baseado em modelos num sistema tutor inteligente para programaçãoo com padrões pedagógicos. Dissertação de mestrado, Instituto de Matemática e Estatística.

Johnson, W. L. and Soloway, E. (1984). "PROUST: Knowledge-based program understanding". In *ICSE '84: Proc. of the 7th intl. conference on Software engineering*, pages 369–380, Piscataway, NJ, USA. IEEE Press.

Mateis, C., Stumptner, M., and Wotawa, F. (2000). "A value-based diagnosis model for java programs". In *In Proc. of the 11th Intl. Workshop on Principles of Diagnosis*.

Mayer, W., Stumptner, M., Wieland, D., and Wotawa, F. (2002). "Observations and results gained from the jade project". In *Proc. of the 13th Intl. Workshop on Principles of Diagnosis*, Semmering, Austria.

Mozetič, I. (1991). "hierarchical model-based diagnosis". *Intl. Journal of Man-Machine Studies*, 35(3):329–362.

Porter, R. and Calder, P. (2003). "A pattern-based problem-solving process for novice programmers". In *ACE '03: Proc. of the 5th Australasian conf. on Computing education*.

Reiter, R. (1987). "A theory of diagnosis from first principles". *Artificial Intelligence*, 32(1):57–95.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Morgan Kaufmann Press.

Winslow, L. E. (1996). "Programming pedagogy – a psychological overview". *SIGCSE Bull.*, 28(3):17–22.