Uma Estratégia para Geração de Dados de Teste

Silvia Regina Vergilio (DINF-UFPR)
José Carlos Maldonado (ICMSC-USP)
Mario Jino (FEE-UNICAMP)

Centro Politecnico - Jardim das Américas
81531-970 - C.P. 19081 - Curitiba - PR
e-mail: silvia@inf.ufpr.br

RESUMO

Teste de software é uma das atividades mais onerosas dentre as de garantia de qualidade de software. Dentro das atividades de teste, a etapa mais difícil de ser automatizada é a de geração de dados de teste para caminhos que cobrem os elementos requeridos por um determinado critério, pois não existe algoritmo tal que, dado um caminho completo qualquer, forneça o conjunto de valores para as variáveis de entrada do programa que executa o caminho dado; não existe algoritmo nem mesmo para dizer se esse conjunto existe, ou seja, se o caminho é executável. Neste trabalho é apresentada uma estratégia para reduzir os efeitos causados por caminhos não executáveis na geração de dados de teste. A estratégia permite em alguns casos, evitar que esforço seja gasto com esses caminhos e também permite reduzir custos, minimizando o número de caminhos a serem executados. Também é apresentada uma maneira de gerar dados de teste para os elementos requeridos pelo teste estrutural, a fim de que eles possam cobrir também certos tipos de erros, detectados por exemplo, por técnicas funcionais. Esses aspectos são discutidos através de exemplos de utilização.

ABSTRACT

Software testing is one of the most expensive activity with respect to software quality. Test data generation in software testing is the most difficult task to be automated, once there is no algorithm to determine a set of values to execute a particular path in a program. Furthermore, there is no algorithm to determine whether this set exists, that is, whether the path is executable or not. This paper presents one strategy to reduce the effects of infeasible paths in test data generation. This strategy for some cases, permits to reduce effort with this paths and to minimize the number of paths executed. Also, describes how the execution of required paths for structural testing methods can reveal type of errors detected by other methods - for example functional testing methods. These aspects are discussed with examples.

1 INTRODUÇÃO

Teste é uma das atividades fundamentais e mais custosas de garantia da qualidade de software. Segundo Pressman [20], em alguns casos o teste pode custar de três a cinco vezes mais que todas as outras atividades do processo de Engenharia de Software e consumir até 50% dos custos de desenvolvimento de programas típicos.

Técnicas de testes requerem a execução do progama com o objetivo de encontrar erros. Se essas técnicas pudessem ser completamente automatizadas, os custos diminuiriam substancialmente. Na aplicação dessas técnicas, a tarefa de geração de dados de teste é a mais difícil e complexa de ser automatizada; sendo esse um problema equivalente ao problema da parada [10].

Dentre as técnicas de teste existentes podemos citar: **Técnica de Teste Baseado em Erros**: está baseada em certos tipos de erros utilizados para derivar casos de teste. Os casos de teste gerados são específicos para mostrar a presença ou ausência desses erros (ex. Análise de Mutantes); **Técnica Funcional**: estabelece casos de teste baseados na especificação e na identificação dos requisitos funcionais (ex. Análise do Valor Limite, Grafos de Causa-Efeito, Teste de Partição); **Técnica Estrutural**: utiliza o código fonte e a particular implementação para estabelecer os casos de teste (ex. Teste Baseado em Fluxo de Controle, Teste Baseado em Fluxo de Dados).

Técnicas estruturais, em geral, requerem que o conjunto de casos de teste T exercite determinados elementos do grafo de fluxo de controle (que podem ser arcos, nós, caminhos) e estabelecem um critério de teste estrutural a ser satisfeito. Satisfazer um critério significa exercitar todos os elementos requeridos por esse critério; neste caso, diz-se que o conjunto de casos de teste T utilizado é adequado em relação ao critério em consideração. Nesse sentido, geração de dados de teste é o processo de identificar dados de entrada para o programa, tal que um critério selecionado seja satisfeito.

Segundo [13] um gerador de dados de teste é uma ferramenta que auxilia o programador na geração de dados de teste para um programa. Na literatura são encontrados alguns tipos de geradores de dados de teste:

Geradores de Dados de Teste Randômicos [3]: os dados de teste são produzidos sem referência ao código ou especificação; intuitivamente parecem ser de pouco valor prático, pois não garantem execução de nenhum critério de teste, que segundo [19], é um ítem, entre outros, a ser satisfeito para se considerar a atividade de teste encerrada.

Geradores de Dados de Teste Para Execução de Caminhos [4], [6], [21], [11], [2], [13]: uma vez estabelecido um conjunto de caminhos a ser executado (que cobre os elementos requeridos por um determinado critério), o gerador deriva dados de entrada que executarão cada caminho do conjunto.

Geradores de Dados de Teste Para Análise de Mutantes [7]: ferramentas criadas com o objetivo de gerar dados de teste baseados em certos tipos de erros, que fazem com que programas mutantes (variações do programa em teste) se comportem diferentemente.

Neste trabalho são de especial interesse os geradores de dados de teste para execução de caminhos, que aceitam como entradas o programa em teste e um critério a ser satisfeito pelo conjunto de dados de teste a ser gerado. A maioria desses geradores de dados de teste utilizam execução simbólica para determinar os dados de entrada [4], [6], [21], [11]. Dado um caminho, que executa um elemento exigido pelo critério, ele será executado simbolicamente e um conjunto de equações envolvendo as variáveis de entrada é criado. A partir de possíveis soluções para esse conjunto serão derivados os dados de teste.

As maiores limitações da execução simbólica são o tratamento de variáveis dos tipos arrays, ponteiros e registros, e o tratamento de chamadas de funções. Korel [13] propõe uma alternativa para essas limitações, referenciada como técnica dinâmica; está baseada na execução real do programa, em análise de fluxo de dados e funções de minimização. No entanto, para os geradores

de dados de teste em geral, o maior problema é a existência de caminhos não executáveis. Um caminho é dito não executável se não existir um conjunto de valores para as variáveis de entrada, variáveis globais e parâmetros do programa que causam a execução do caminho.

Um critério de teste estabelece elementos do grafo de fluxo de controle a serem exercitados por um conjunto de caminhos. Dados de teste para esses caminhos precisam ser gerados. Sendo a tarefa de geração de dados de teste tão custosa, para reduzir o esforço gasto nessa tarefa, o ideal seria que o conjunto selecionado contivesse um número mínimo de caminhos, que fossem deteminados os elementos não executáveis dentre os elementos requeridos pelo critéiro, e que fossem selecionados somente caminhos executáveis.

Determinar se um caminho é ou não executável é uma questão indecidível, ou seja, não existe algoritmo tal que, dado um caminho completo qualquer, forneça o conjunto de valores que causam a execução do caminho; não existe algoritmo nem mesmo para decidir se esse conjunto existe [8]. As dificuldades e o custo para geração de dados de teste aumentam consideravelmente, pois enorme quantidade de esforço e tempo são gastos na tentativa de gerar dados de teste para esses caminhos; e a maioria dos programas até mesmo os bem formulados contém caminhos não executáveis. Num estudo com 27 programas extraídos de [12], 24 deles apresentam caminhos não executáveis; de um total de 2371 caminhos analisados, 1338(56.5%) são não executáveis [24],[25].

Esforços têm sido conduzidos no sentido de caracterizar, prever e determinar caminhos não executáveis [8],[18],[24],[25], para facilitar a atividade de teste. Malevris et al [18] objetivando prever a não executabilidade estudam a relação entre o número de predicados de um caminho e sua executabilidade em um conjunto de rotinas e evidenciam a validade da hipótese: quanto maior o número de predicados de um caminho, maior a probabilidade de ele ser não executável. O conjunto de caminhos analisados tinha o objetivo de satisfazer o Critério Estrutural Todos-Ramos e os elementos explorados foram os LCSAJ's ("linear code sequence and jump").

Em [24],[25], estão os resultados obtidos durante a condução de um "benchmark" (conjunto de rotinas utilizadas por Weyuker [27]) com respeito a não executabilidade de caminhos. Esse "benchmark" foi conduzido utilizando-se uma ferramenta de testes denominada POKE-TOOL [5] que tem como objetivo determinar caminhos e associações requeridas e avaliar um conjunto de casos de teste com relação aos Critérios Potenciais Usos [16],[17].

Um dos resultados mais significativos está em [26]; neste trabalho a validade da hipótese de Malevris et al [18] é explorada no contexto descrito. Modelos estatísticos são apresentados e evidenciam a validade dessa hipótese para os programas do "benchmark" conduzido, no contexto de Teste Baseado em Fluxo de Dados. A hipótese é então utilizada na elaboração de uma estratégia para gerar dados de teste para os critéiros baseados em fluxo de dados; entre dois caminhos completos possíveis, que levam à execução de um ou mais elementos requeridos, é selecionado o caminho com menor número de predicados, pois esse tem maior probabilidade de ser executável. A estratégia acima é descrita neste trabalho. Ela tem por objetivo reduzir os efeitos causados pelos caminhos não executáveis nas atividades de teste e visa diminuir os custos destas atividades. Permitindo:

- Identificar alguns elementos não executáveis
- Reduzir o número de caminhos não executáveis selecionados para cobrir os elementos requeridos
- Que os caminhos executados possam revelar outros tipos de erros além dos tipos detectados por técnicas estruturais

Primeiramente será dada uma breve descrição sobre o contexto dos critérios baseados em fluxo de dados e justificativas estatísticas comprovando a validade da hipótese de Malevris. Depois será dado um exemplo de utilização da estratégia, utilizando-se os critérios Potenciais Usos e a ferramenta de teste POKE-TOOL.

2 Influência do Número de Predicados na Executabilidade de Um Caminho

Recentemente foram introduzidos critérios que utilizam informações sobre o fluxo de dados para o estabelecimento de requisitos de teste, denominados Critérios Estruturais Baseados em Fluxo de Dados [9],[14],[22],[23],[16]. Esses critérios exploram associações entre pontos em que ocorre a definição de variáveis e pontos onde essas variáveis são utilizadas ou potencialmente utilizadas, como é o caso dos Critérios Potenciais Usos.

Os Critérios Potenciais Usos básicos - Critério Todos-Potenciais-Usos, Critério Todos-Potenciais-Usos/Du, Critério Todos-Potenciais-Du-Caminhos - foram definidos em [16],[17] e estão baseados no conceito de potencial uso. Eles requerem que caminhos livres de definição em relação a qualquer nó i, com definição de variável, e a qualquer variável x, definida em i, sejam executados, independentemente de ocorrer um uso de x nesses caminhos. Assim, define-se uma **potencial associação** (i,j,x) ou (i,(j,k),x), entre o nó i e o nó j ou arco (j,k) do grafo de fluxo de controle, se existir um caminho livre de definição com respeito a x (c.r.a x) de i para j; define-se um **potencial du-caminho** (n1,...,nj,nk) c.r.a x do nó n1 para nk ou para o arco (nj,nk), se (n1,...,nj,nk) é livre de definição c.r.a x e (n1,...,nj) é livre de laços e no nó n1 ocorre uma definição de x.

Maldonado [17] introduziu a notação [i,(j,k),{v1,...vn}] para representar o conjunto de associações [i,(j,k),v1],...[i,(j,k),vn], indicando que existe pelo menos um caminho livre de definição c.r.a v1,..vn do nó i para o arco (j,k); observe-se que poderão existir outros caminhos livres de definição c.r.a algumas das variáveis v1,..vn, mas não simultaneamente a todas elas.

A aplicação dos Critérios Potenciais Usos é facilitada pela disponibilidade de uma ferramenta de teste - POKETOOL [5]. Esta ferramenta, entre outras características, determina os elementos requeridos e faz a avaliação da adequação de um conjunto de casos de teste, T, em relação a esses critérios.

Para verificar o comportamento e a aplicação dos Critérios Potenciais Usos em programas reais e estudar a ocorrência e a influência de caminhos não executáveis, foi conduzido um "benchmark", utilizando-se a POKE-TOOL. As principais causas de não executabilidade encontradas durante a análise dos caminhos são discutidas em [24],[25]. Durante essa análise, as principais dificuldades encontradas foram a avaliação dos predicados compostos e o estudo da executabilidade de caminhos com um número grande de predicados. Isso foi uma motivação para realizar estudos para comprovação da validade da hipótese de Malevris para as rotinas do "benchmark" conduzido.

Apesar das rotinas do "benchmark" serem programas bem formulados, o número de caminhos e associações não executáveis encontrado foi maior do que o esperado. Apenas três das 27 rotinas não apresentaram caminhos não executáveis. E além disso, identificar manualmente a executabilidade dos caminhos e associações foi uma tarefa tediosa que consumiu muito esforço e tempo. Para validar a hipótese de Malevris foi também calculado o número de predicados dos caminhos executados (executáveis) e dos que não puderam ser executados (não executáveis).

Tabela 2.1. Número de Predicados Contidos nos Potenciais-du-Caminhos das Rotinas do "Benchmark" Excluindo-se as Unidades Recursivas

nprd	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	tot
nex	0	19	40	37	83	109	190	228	105	109	130	111	57	31	24	40	14	5	6	1338
exc	19	104	104	101	115	106	117	96	50	44	61	33	25	16	10	17	8	5	2	1033
tot	19	123	144	138	198	215	307	324	155	153	191	144	82	47	34	57	22	10	8	2371

Deve-se ressaltar que os estudos foram realizados excluindo-se as duas unidades recursivas. Pode-se ver na Tabela 2.1 que 56.5% dos caminhos analisados são não executáveis; de um total de 47 caminhos com 13 predicados, 16 são executáveis e 31 são não executáveis.

Para estudar o relacionamento entre o número de predicados de um potencial-du-caminho e a sua executabilidade, como feito por Malevris et al, explorou-se a hipótese H_0 — não existe uma associação entre a executabilidade de um caminho e o número de predicados nele contido, ou seja, existem proporções iguais de caminhos executáveis para qualquer q>1 — aplicando-se um teste χ^2 e utilizando-se os dados da Tabela 2.1, foram obtidos $\chi^2 = 323.70$ com $v = 17$ (número de graus de liberdade). Como $\chi^2_v(0.005) = 35.718$, tem-se que H_0 é rejeitada, concluindo-se, portanto, que existe uma associação entre o número de predicados do caminho e sua executabilidade; ou seja, a proporção de caminhos executáveis para q predicados não é a mesma para qualquer valor de q e deve existir alguma forma de dependência entre a executabilidade de um caminho e o número de predicados q nele contido.

Foram explorados modelos polinomiais e modelos exponenciais, a exemplo de Malevris [18]; embora uma função polinomial em q possa ser um bom modelo para $1 \leq q \leq 18$, quando q aumenta, a função aumenta ou diminui monotonicamente sem limite e, por isso, os modelos exponenciais são de maior interesse. Pôde-se observar que, para $1 < q \leq 9$, excluindo-se ou não as unidades recursivas, a hipótese de Malevris de que, quanto maior o número de predicados em um caminho menor a probabilidade de ele ser executável, é também válida para potenciais-du-caminhos, para o "benchmark" considerado. Um dos modelos exponenciais obtidos foi $p = 1.018\ e^{-0.149q}$, com $r^2 = 95.0$. Esse modelo e outros são apresentados detalhadamente em [24]. Desta forma, esse modelo exponencial expressa o relacionamento entre os valores apresentados e mostra, comprovando a hipótese de Malevris que, quando aumenta o número de predicados, o número de caminhos executáveis tende a diminuir.

Para determinar associações e caminhos não executáveis foram incorporadas à POKE-TOOL rotinas que implementam a heurística proposta por Frankl [8], além de algumas extensões e facilidades adicionais para tratamento de não executabilidade [24],[25]. A heurística está baseada em técnicas de execução simbólica e análise de fluxo de dados. Para reduzir as limitações existentes na execução simbólica foi implementada uma interface com o usuário, cuja participação, em alguns casos, é fundamental.

3 A Estratégia

Nesta seção é apresentada uma estratégia de geração de dados de teste para cobrir os elementos que, em geral, são exigidos pelos critérios baseados em fluxo de dados [9],[14],[22],[23],[17]: associações e caminhos. A estratégia apresentada é independente do critério utilizado; quanto mais rigoroso for este critério, maior o número e a variedade dos tipos de erros encontrados pelo conjunto de dados de teste gerado.

Como gerar dados de teste para cobrir uma associação?

1. *Dada uma lista de associações exigidas pelo critério, qual associação selecionar?* Seria desejável gerar dados de teste somente para associações executáveis. A heurística de Frankl [8] e as extensões propostas [24] podem ser utilizadas para determinar, em alguns casos, associações não executáveis e evitar que esforço seja gasto com essas associações.

Entre dois conjuntos c1: (i,(j,k),V1) e c2: (i,(j,k),V2), (analogamente para (i,j,V1) e (i,j,V2)), sendo V1 e V2 conjuntos de variáveis tal que $V2 \subset V1$, selecionar o conjunto c1, pois se c1 for coberto, c2 também o será, uma vez que todo caminho livre de definição com respeito às variáveis de V1 é também livre de definição com respeito às variáveis de V2, mas o contrário nem sempre acontece.

Para determinar quais os caminhos livres de definição c.r. às variáveis da associação pode-se utilizar o grafo(i). O grafo(i) é um grafo gerado pela POKE-TOOL, contendo todos os potenciais-du-caminhos com início no nó i, tal que i possui definição de variável. Cada nó k possui um conjunto, notação: deff(k), que contém as variáveis definidas em i, mas não redefinidas num caminho de i para k;

2. *Qual caminho cobre uma associação?* São caminhos completos que contêm subcaminhos livres de definição com respeito às variáveis da associação de i para (j,k) (ou j). Um algoritmo para geração de caminhos completos a partir de caminhos do grafo(i) é dado em [15]; esse algoritmo está sendo implementado e será futuramente incorporado à POKE-TOOL [1].

3. *Qual caminho completo selecionar?* Entre os caminhos completos determinados existem caminhos não executáveis. Escolher o caminho completo com menor número de predicados; isso visa reduzir o esforço pois, baseando-se em estudos de [18],[24], entre todos os caminhos, o de menor número de predicados tem a maior probabilidade de ser executável.

4. *Como sensibilizar o caminho selecionado?* Propõe-se o uso de Geração de Dados de Teste baseada em Restrições no contexto de Critérios Baseados em Fluxo de Dados. Essas restrições descrevem casos de teste projetados para encontrar certos tipos de erros. A idéia é particionar o domínio de entrada em sub-domínios (Teste Baseado em Domínios [28]); cada sub-domínio corresponde a execução de um caminho em particular. Propõe-se utilizar pontos do sub-domínio que resolvem as restrições (Teste Baseado em Restrições [7]). Nesse sentido, segundo [19] um grande número de erros tende a ocorrer nos limites do domínio de entrada, fundamento da técnica conhecida como Análise do Valor Limite. Os limites de cada sub-domínio poderão ser exercitados a partir de restrições que, quando resolvidas, geram casos de teste com alta probabilidade de encontrar esses erros.

Como gerar dados de teste para cobrir um caminho?

Dado um caminho exigido pelo critério, gerar um caminho completo e iniciar no Passo 3. Se o caminho for completo, iniciar no Passo 4.

Essa estratégia pode ser utilizada analogamente para outros critérios de teste estrutural. Por exemplo, considerar o número de predicados dos caminhos completos selecionados para satisfazer o Critério Todos-Ramos que, apesar de ser o critério de teste estrutural menos exigente, é muito utilizado. Deve-se ressaltar que os Critérios Potenciais Usos, utilizados para ilustrar a estratégia, incluem alguns critérios de teste estrutural baseados em fluxo de dados e além disso, nenhum outro critério baseado em fluxo de dados inclui os Critérios Potenciais Usos. Esta relação de inclusão é dada por uma ordem parcial validada em [17].

4 Utilizando a Estratégia

Ilustra-se a seguir a utilização da estratégia proposta para um programa extraído do "benchmark" (Figura 4.1). São utilizadas as facilidades para tratamento de não executabilidade da ferramenta POKE-TOOL [24], [25]. A estratégia proposta facilita a geração de dados de teste e possibilita a redução do custo dessa atividade, pois permite:

- Determinar elementos requeridos não executáveis, evitando gastar esforço para cobrir esses elementos.

Por exemplo, na Tabela 4.1 encontram-se as associações requeridas pelos critérios Potenciais Usos e geradas pela POKE-TOOL, para o programa da Figura 4.1. A heurística de Frankl e extensões implementadas pelas rotinas de tratamento de não executabilidade da POKE-TOOL, podem nesse caso determinar automaticamente todas as associações não executáveis (22% das associações requeridas), assinaladas com um (*).

- Minimizar o número de caminhos a serem executados.

Segundo o Passo 1, isso é possível. Por exemplo, para gerar dados de teste para cobrir as associações 14, 15, 16 (Tabela 4.1), a associação número 14 deverá ser considerada primeiro, pois todo caminho livre de definição c.r. às variáveis {curln,stat,done} é livre de definição c.r.a {curln,stat} e c.r.a {curln,done} e cobrirá também as associações 15 e 16. Se a associação 14 não fosse selecionada primeiro, poderia ser necessária, no pior caso, a execução de três caminhos.

- Selecionar caminhos com maior probabilidade de serem executáveis.

Isso é possível escolhendo entre os caminhos candidatos a cobrir os elementos requeridos, o caminho com o menor número de predicados, baseando-se nos estudos apresentados na Seção 2.

Novamente, para o programa da Figura 4.1, e tomando-se a associação <5,(4,14),{inline}>, tem-se, utilizando-se o grafo(5) gerado pela POKE-TOOL (ver Figura 4.2), os seguintes potenciais-du-caminhos: 5 6 13 4 14, 5 7 9 10 11 12 13 4 14, 5 7 9 11 12 13 4 14, e 5 7 8 12 13 4 14, livres de definição c.r.a inline. Considerando-se os caminhos através do laço do nó 4 (extensões a ciclo para esses potenciais du-caminhos [24]), tem-se o conjunto de caminhos candidatos a cobrir a associação. Uma vez que, para todos os caminhos completos gerados o laço do nó 4 deve ser considerado, tomar-se-á como caminho com menor número de predicados através do laço do nó 4 aquele que não percorre o laço. Os caminhos então diferem por causa das diferentes maneiras de ir do nó 5 ao arco (4,14), dadas pelo grafo(5). Assim, o caminho completo escolhido é 1 3 4 5 6 13 4 14 15 16. Note que pode-se evitar que o caminho não executável 1 3 4 5 7 9 11 12 4 14 15 16, um dos que possui maior número de predicados, seja escolhido.

- Complementar os tipos de erros encontrados pela técnica estrutural e identificar também tipos de erros cobertos por técnicas funcionais (Análise do Valor Limite) e técnicas baseada em erros (Teste Baseado em Restrições).

Seja o programa da Figura 4.3. Para o caminho 1 2 4 6 requerido pelos Critérios Potenciais Usos, obtém-se o domínio de entrada correspondente a área pontilhada (ilimitada superiormente), no gráfico da Figura 4.4, que é determinado pelos predicados ao longo deste caminho.

$$\begin{cases} J+1 \leq K \\ K-(J+1) > -1 \end{cases}$$

sendo J e K valores iniciais para as variáveis j e k respectivamente.

O caso de teste (J=1, K=5), pertence ao domínio correspondente ao caminho 1 2 4 6, e é solução do sistema de equações dado acima, portanto poderia ser derivado num processo de execução simbólica. Mas, suponha que o programa possui um erro no nó 1, e que o comando correto é j = j + i. O caso de teste (J=1, K=5) executa o caminho 1 2 4 6 como desejado, mas não revela o erro existente. (Ver Tabela 4.2)

Utilizando-se Análise do Valor Limite e Teste Baseado em Restrições como proposto no Passo 4, toma-se como caso de teste para executar o caminho 1 2 4 6, pontos no limite do domínio, projetados para encontrar esse tipo de erro e que satisfaçam a seguinte restrição: (j+1 $\leq$ k) $\neq$ (j+i $\leq$ k), ou seja (j+1 $\leq$ k) $\neq$ (j+2 $\leq$ k).

A solução poderá ser então (J=0, K=1), pontos nos limites do domínio, e que satisfazem a restrição (0+1 $\leq$ 1) $\neq$ (0+2 $\leq$ 1), e como pode ser visto na Tabela 4.3 revelam o erro. Note que para (J=1, K=5): (1+1 $\leq$ 5) = (1+2 $\leq$ 5) = TRUE.

Portanto o caso de teste (J=0, K=1), tem alta probabilidade de revelar um erro que ocorre nos limites do domínio de entrada do caminho 1 2 4 6, e mais, mostrar a presença (ou ausência de um erro típico em programação: a substituição de 1 por i. Assim poderia ser feito para outros erros específicos.

```
stcode append(int line, int glob)
  {
1   char inline[MAXSTR];
1   stcode stat, int done;
1   if (glob)
2     stat=ERR;
3     else
3     {
3     curln=line;stat=OK;done=0;
4     while((!done)&&(stat==OK))
4     {
5       if(!getline(inline,stdin,MAXSTR))
6         status=ENDDATA;
7       else
7       {
7         if((inline[0]==PERIOD)&&
            (inline[13]==NEWLINE))
8           done=1;
9         else
9         {
9           if(puttxt(inline)==ERR)
10            stat=ERR;
11        }
12      }
13    }
14  }
15  return stat;
16 }
```

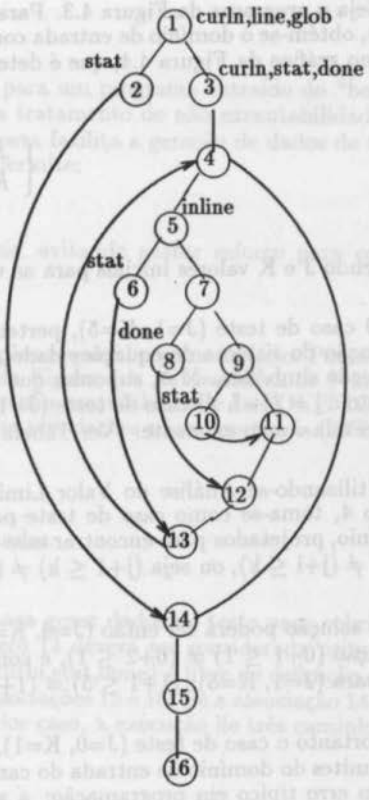
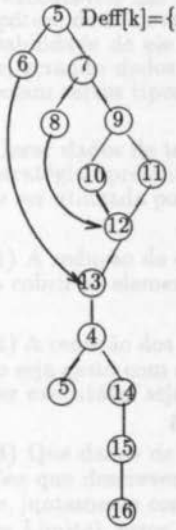


Figura 4.1: Rotina Append-Código Fonte e Grafo/Def

Tabela 4.1. Associações - Critérios Todos-Pot-Usos e Pot-Usos/Du

1)<1,(4,14),{line,glob}>	18)<5,(9,11),{inline}>	35)<8,(6,13),{done}> *
2)<1,(9,11),{line,glob}>	19)<5,(9,10),{inline}>	36)<8,(5,6),{done}>
3)<1,(9,10),{line,glob}>	20)<5,(7,8),{inline}>	37)<9,(9,11),{inline}>
4)<1,(7,8),{line,glob}>	21)<5,(4,14),{inline}>	38)<9,(4,14),{inline}>
5)<1,(13,4),{line,glob}>	22)<5,(4,5),{inline}>	39)<9,(7,9),{inline}>
6)<1,(5,6),{line,glob}>	23)<5,(5,6),{inline}>	40)<9,(8,12),{inline}>
7)<1,(1,3),{curln,line,glob }>	24)<6,(4,14),{stat}>*	41)<9,(6,13),{inline}>
8)<1,(1,2),{curln,line,glob}>	25)<6,(9,11),{stat}>*	42)<9,(6,13),{inline}>
9)<2,(,),{stat}>	26)<6,(9,10),{stat}>*	43)<9,(5,6),{inline}>
10)<3,(4,14),{curln,stat,done}>	27)<6,(12,13),{stat}>	44)<9,(9,10),{inline}>
11)<3,(9,11),{curln,stat,done}>	28)<6,(7,8),{stat}>	45)<10,(4,14),{stat}>*
12)<3,(9,10),{curln,stat,done}>	29)<6,(5,6),{stat}>	46)<10,(9,11),{stat}>*
13)<3,(7,8),{curln,stat,done}>	30)<8,(4,14),{done}>	47)<10,(9,10),{stat}>*
14)<3,(13,4),{curln,stat,done}>	31)<8,(9,11),{done}>	48)<10,(8,12),{stat}>*
15)<3,(13,4),{curln,stat}>	32)<8,(11,12),{done}>	49)<10,(7,8),{stat}>*
16)<3,(13,4),{curln,done}>	33)<8,(9,10),{done}>*	50)<10,(5,6),{stat}>
17)<3,(5,6),{curln,stat,done}>	34)<8,(7,8),{done}>*	



Potencias-du-caminhos para [5,(4,14),inline]	número de predicados
5 6 13 4 14	3
5 7 8 12 13 4 14	5
5 7 9 10 12 13 4 14	6
5 7 9 11 12 13 4 14	6

Figura 4.2 Grafo(5) e Pot-du-caminhos para Assoc [5,(4,14),inline]

```
int troca()
1 { int i,j,k,m;
1 leitura(&j,&k);
1 i=2; j=j+1;
1 if(j<=k)
2 { j=k-j;
2 m=0;}
3 else
4 { j=j-k;
4 m=1;}
4 if(j<-1)
5 { m=2;}
6 return m;
6 }
```

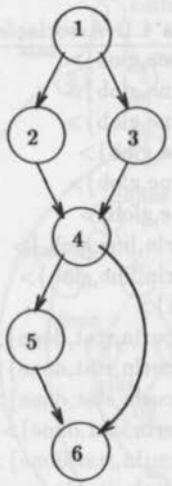


Figura 4.3 Exemplificando o Passo 4

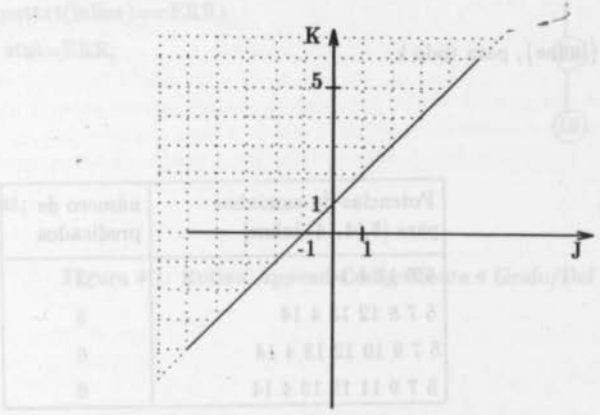


Figura 4.4. Domínio de Entrada para o caminho 1 2 4 6

Tabela 4.2. Resultado da Execução do Caso de Teste (J=1, K=5)

		caminho executado	resultado esperado
programa real	j = j + 1	1 2 4 6	m = 0
programa correto	j = j + i	1 2 4 6	m = 0

Tabela 4.3. Resultado da Execução do Caso de Teste (J=0, K=1)

		caminho executado	resultado esperado
programa real	j = j + 1	1 2 4 6	m = 0
programa correto	j = j + i	1 3 4 6	m = 1

5 Conclusões

Os resultados apresentados na Seção 2, demonstram que na prática, a maioria dos programas, até mesmo os bem formulados, possui caminhos não executáveis. Isso traz muitos problemas para automatizar as atividades de geração de dados de teste, já que não existe algoritmo para determinar a executabilidade de um caminho.

A estratégia de geração de dados de teste visa minimizar os efeitos causados por caminhos não executáveis nas atividades de teste. Ela foi derivada de estudos conduzidos para validar a hipótese de Malevris de que quanto maior o número de predicados de um caminho maior a probabilidade de ele ser não executável. Também permite a utilização de outras técnicas de teste, gerando dados que executam os caminhos selecionados pelas técnicas estruturais e que detectam certos tipos de erros comuns em programação.

Gerar dados de teste que executem um caminho dado é uma tarefa muito difícil e custosa. A estratégia apresentada evita que esforço seja gasto na realização dessa tarefa. A estratégia pode ser utilizada por geradores de dados de teste para execução de caminhos e permite:

1) A redução de esforço de geração de dados de teste, minimizando o número de caminhos para cobrir os elementos requeridos por um dado critério de teste.

2) A redução dos efeitos causados por caminhos não executáveis pois, pode-se evitar que esforço seja gasto com alguns elementos não executáveis e que caminhos com maior probabilidade de ser executável sejam selecionados.

3) Que dados de teste gerados para cobrir os elementos requeridos, satisfaçam certas restrições que descrevem certos tipos de erros, possibilitando a utilização de outras técnicas de teste, juntamente com a técnica estrutural; erros nos limites do domínio de entrada (Análise do Valor Limite), erros que façam programas mutantes se comportarem diferentemente (Análise de Mutantes).

Pretende-se explorar a validade dessa estratégia para todo o "benchmark" e inclusive compará-la com outras existentes. A validade da abordagem proposta no Passo 4 deverá ser explorada num conjunto de programas (estudo de casos) e o suporte utilizado nessa tarefa deverá ser desenvolvido.

Referências

[1] Bernardes, M.C., Implementação de Rotinas para Apoio à Geração de Dados de Teste, Relatório de Iniciação Científica, DCA/FEE/UNICAMP, Campinas, S.P., Janeiro, 1993.

[2] Bicevks, J.; Borzovs, J.;Shaujums, U.; Zarins, A.; Miller, E., "SMOTL - A System to Construct Samples for Data Program Debugging", IEEE Trans. on Software Eng., 5(1), 60-66, Jan., 1979.

[3] Bird, D.; Munoz, C., "Automatic Generation of Random Self-Checking Test Cases", IBM Syst. J., 22(3), 229-245, 1983.

[4] Boyer, R.S.; Elspas, B.; Levitt, K.N., "Select - A Formal System for Testing and Debugging Programs by Symbolic Execution", SIGPLAN Notices, 10(6), 234-245, June, 1975.

[5] Chaim, M.L., POKE-TOOL - Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados. *Tese de Mestrado*, DCA/FEE/UNICAMP - Campinas, SP, Abril 1991.

[6] Clarke, L., "A System to Generate Test Data and Symbolic Execute Programs", IEEE Trans. on Software Eng., SE-2(3), 215-222, Sept., 1976.

[7] DeMillo, R.A.; Offutt J.; "Constraint-Based Automatic Test Data Generation", IEEE Trans. on Software Eng., 17(9), Sept., 1991.

[8] Frankl, F.G., The Use of Data Flow Information for Selection and Evaluation of Software Test Data, PhD Dissertation, New York, Oct., 1987.

[9] Herman, P.M., "Flow Analysis Approach to Program Testing", The Australian Computer Programs, Proc. 8th ICSE, London, UK, 1985.

[10] Howden, W., "Methodology for The Generation of Program Test Data", IEEE Trans. on Comp., 24(5), May, 1975.

[11] Howden, W., "Symbolic Testing and The DISSECT Symbolic Evaluation", IEEE Trans.on Software Eng., SE-4(4), 266-278, July, 1977.

[12] Kernighan, B.W.; Plauger, P.J., Software Tools in Pascal. Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.

[13] Korel, B., "Automated Software Test Data Generation", IEEE Trans.on Software Eng., 6(8), 870-879, August, 1990.

[14] Laski, J.W.; Korel, B., "A Data Flow Oriented Program Testing Strategy", IEEE Trans.on Software Eng., 9(3), 347-354, May, 1983.

[15] Maldonado, J.C; Chaim, M.L.; Jino, M., Resultados do Estudo de uma Família de Critérios de Teste de Programas Baseada em Fluxo de Dados, Relatório Técnico - DCA/FEE/UNICAMP - RT/DCA-001/88 - Campinas, SP, Set., 1988.

[16] Maldonado, J.C; Chaim, M.L.; Jino, M., "Seleção de Casos de Teste Baseada nos Critérios Potenciais Usos", II Simpósio Brasileiro de Engenharia de Software, Canela, RS, Out., 1988.

[17] Maldonado, J.C., Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software. *Tese de Doutorado*, DCA/FEE/ UNICAMP - Campinas, SP, Julho 1991.

[18] Malevris, N.; Yates, D.F.; Veevers, A., "Predictive Metric for Likely Feasibility of Program Paths", Information and Software Technology, 32(2), March 1990.

[19] Myers, G.J., The Art of Software Testing, Wiley, 1979.

[20] Pressman, R.B., Software Engineering: a Practitioner's Approach, Third Edition, New York, McGraw-Hill, 1992.

[21] Ramamoorthy, S.; Ho, F.S.; Chen, W.T., "On The Automated Generation of Program Test Data, IEEE Tran. on Software Eng., SE-2(4), 293-300, Dec., 1976.

[22] Rapps, S.; Weyuker, E.J., "Data Flow Analysis Techniques for Test Data Selection", *in* Proc. Int. Conf. Software Engineering, Tokyo, Sept., 1982.

[23] Ural, U.; Yang, B., "A Structural Test Selection Criterion", Information Processing Letters, Jul., 1988.

[24] Vergilio, S.R., Caminhos Não Executáveis: Caracterização, Previsão e Determinação para Suporte ao Teste de Programas. *Tese de Mestrado*, DCA/FEE/UNICAMP, Campinas, SP, Jan. 1992.

[25] Vergilio, S.R.; Maldonado, J.C.; Jino, M. "Caminhos Não Executáveis na Automação das Atividades de Teste", VI Simpósio Brasileiro de Engenharia de Software, Gramado, RS, Nov. 1992.

[26] Vergilio, S.R.; Maldonado, J.C; Jino, M., "Influência do Número de Predicados na Executabilidade de Um Caminho no Contexto de Teste Baseado em Fluxo de Dados", XIX Conferencia Latinoamericana de Informatica, Buenos Aires, Argentina, Março, 1993.

[27] Weyuker, E.J., "The Cost of Data Flow Testing: An Empirical Study", IEEE Trans. on Software Eng. SE-16(2), 12-18, Feb., 1988.

[28] White, L.J.; Cohen, E.I., "A Domain Strategy for Computer Program Testing", IEEE Trans. on Software Eng., 6(3), 247-257, May, 1980.