CAMINHOS NÃO EXECUTÁVEIS NA AUTOMAÇÃO DAS ATIVIDADES DE TESTE

Silvia R. VERGILIO
José C. MALDONADO
Mário JINO

Departamento de Computação e Automação Industrial,
Universidade Estadual de Campinas - UNICAMP,
C.P. 6101, 13081 Campinas, SP

RESUMO

No caso geral, não é possível determinar (é indecidível) se um dado caminho através de um programa é ou não executável; a èxistência de caminhos não executáveis traz muitos problemas para a automatização das atividades de teste. Esses problemas são discutidos, mais especificamente dentro do contexto da ferramenta de testes POKE-TOOL, que apóia a aplicação dos Critérios Potenciais Usos - uma família de critérios de teste baseados em análise de fluxo de dados. Um "benchmark" foi realizado utilizando-se a POKE-TOOL. As principais causas de não executabilidade dos caminhos encontrados nas rotinas do "benchmark são descritas. Foi estudado o relacionamento entre o número de predicados de um caminho e sua executabilidade. Outro resultado do "benchmark" foi ressaltar a relevância da heurística proposta por Frankl [FRA87] para determinação de não executabilidade. Adicionalmente, são descritas duas extensões para esta heurística e outros recursos para tratamento de caminhos não executáveis (como exemplo a eliminação de padrões de não executabilidade). Esses recursos foram incorporados à ferramenta POKE-TOOL, com o objetivo de reduzir o custo e esforço das atividades de teste.

ABSTRACT

It is undecidable whether any given program path is feasible or not. The main problems introduced by infeasible paths in program testing are discussed. The results presented are taken from benchmarking Potential Uses Criteria --- a family of data-flow based criteria --- using the testing tool named POKE-TOOL, which supports the application of these criteria. The main causes for infeasibility of the benchmark's program paths are described. Correlation between infeasibility and the number of predicates in program paths is also explored. The benchmark pointed out the relevance of the Frankl's heuristics [FRA87] for identification of infeasible paths. The heuristics and two proposed extensions, as well as, other facilities to deal with such paths, such as, elimination of infeasible patterns, are described. These facilities, incorporated into POKE-TOOL, contribute to reduce testing activities cost and effort.

1 - INTRODUÇÃO

O desenvolvimento de software está sujeito a erros de vários tipos. Esses erros são, em geral, erros humanos originados na maioria das vezes por falhas de transformação e/ou comunicação nas diversas fases de desenvolvimento do software. Na engenharia de software, atividades de teste e validação são fundamentais para garantir a qualidade do software. A falta de tempo e de recursos humanos capacitados, agravados pela não disponibilidade de ferramentas adequadas, são os principais problemas enfrentados pelas equipes de teste. Acrescente-se ainda, que as atividades de teste consomem, cerca de 50% do tempo e custo do desenvolvimento de software [MYE79].

Muitas técnicas surgiram nas últimas décadas para projetar casos de teste efetivos, visando oferecer uma maneira sistemática que determine qual o subconjunto de todos os possíveis casos de teste capaz de detectar a maioria dos erros, com o mínimo de tempo e esforço. Recentemente, foram introduzidos critérios estruturais baseados em análise de fluxo de dados para a seleção de casos de teste [RAP85, FRA86, MAL88, MAL91a]. A idéia básica é executar os caminhos que estabelecem associações entre o ponto onde uma variável do programa é definida e o ponto onde ela é utilizada.

Em geral, existem elementos não executáveis; os critérios determinam a escolha dos elementos que devem ser exercitados baseados apenas na sintaxe do programa. Um caminho de um programa é dito executável se existe alguma atribuição de valores das variáveis de entrada, variáveis globais e parâmetros do programa que causam a sua execução; caso contrário, ele é dito não-executável [FRA87]. A presença de caminhos não executáveis perturba consideravelmente a ordem parcial entre os critérios; por exemplo, alguns dos critérios, na presença de caminhos não executáveis, deixam de incluir o critério todos os ramos e/ou incluir um uso de todo resultado computacional. Outro aspecto afetado pela presença de caminhos não executáveis é o do encerramento das atividades de teste; Myers [MYE79] considera um critério de teste como um item entre outros a serem satisfeitos para se considerar a atividade de teste encerrada; para que um critério seja satisfeito é necessário exercitar todos os elementos por ele requeridos.

A existência de caminhos não executáveis impede a geração de dados de teste para execução de todos os caminhos exigidos pelo critério e, consequentemente, a satisfação do critério dado. Na prática, para a maioria dos programas, até mesmo os bem formulados, um número relativamente grande de caminhos não executáveis é encontrado. O custo das atividades de teste aumenta consideravelmente devido à não executabilidade

de caminhos [MAY90] pois, embora na maioria das vezes seja fácil para o ser humano identificar se um caminho é ou não executável, não existe algoritmo que decida a sua executabilidade; ou seja, é indecidível se para um dado programa P existe um conjunto de casos de teste T que satisfaz C, devido ao aspecto da executabilidade de caminhos [FRA86, FRA87].

Segundo Malevris [MALE90] a existência de caminhos não executáveis foi documentada por volta de 1972 e, surpreendentemente, poucos autores têm-se dedicado aos problemas relativos a caminhos não executáveis. Entre os trabalhos existentes na literatura, destacam-se: os estudos de Hedley e Hennel [HED85] sobre as principais causas de não executabilidade de caminhos; o trabalho de Malevris [MALE90] para determinar uma heurística para prever não executabilidade; e a heurística proposta por Frankl [FRA87] para determinar associações não executáveis.

Para dar suporte à utilização dos critérios Potenciais Usos foi implementada uma ferramenta de teste, denominada POKE-TOOL [MAL91a, CHA91]. Os critérios Potenciais Usos foram definidos por Maldonado, Chaim e Jino [MAL88, MAL91a] e exigem a execução de caminhos livres de definição, a partir da definição de uma determinada variável do programa, independentemente da ocorrência de um uso dessa variável nesses caminhos. Esta ferramenta está operacional para programas escritos na linguagem C e foi desenvolvida para ambientes do tipo PC. Para verificar o comportamento e a aplicação dos critérios Potenciais Usos em programas reais, foi conduzido um "benchmark" proposto por Weyuker [WEY90], utilizando-se a POKE-TOOL. Esse "benchmark" consiste de 29 rotinas extraídas do livro "Software Tools in Pascal", de Kernighan & Plauger [KER81]. Uma descrição mais detalhada da aplicação e dos objetivos do "benchmark" é dada em [MAL91a].

Apesar das rotinas do "benchmark" serem programas bem formulados, o número de caminhos e associações não executáveis encontrado foi maior que o esperado. Além disto, identificar manualmente a executabilidade dos caminhos e associações foi uma tarefa tediosa que consumiu muito esforço e tempo.

O objetivo deste trabalho é abordar três aspectos relacionados a caminhos não executáveis: caracterização, previsão e determinação. Na Seção 2, é dada uma síntese dos principais resultados obtidos durante a condução do "benchmark"; é apresentada uma caracterização das principais causas de não executabilidade do caminhos, e determina-se o relacionamento entre o número de predicados de um caminho e sua executabilidade. Na Seção 3 é apresentada a heurística proposta por Frankl para determinação de não executabilidade; são propostas duas extensões para essa heurís-

tica; adicionalmente são discutidos os recursos que foram incorporados à POKE-TOOL para tratamento de não executabilidade. As conclusões são apresentadas na Seção 4.

2 - RESULTADOS DA APLICAÇÃO DE UM "BENCHMARK"

A aplicação do "benchmark" consistiu de duas atividades principais: a elaboração dos conjuntos de casos de teste e a análise da adequação desses conjuntos em relação aos critérios Potenciais Usos Básicos -todos potenciais-usos, todos potenciais-usos/du e todos potenciais du-caminhos. São apresentados os principais resultados do "benchmark", mais particularmente relativos à caracterização e previsão de caminhos não executáveis; outros resultados estão em [MAL91a,MAL91b,VER92].

2.1 - Previsão de Não Executabilidade

Para cada um dos programas que compõem o "benchmark", foram determinados os seguintes dados (características do programa): número de comandos de decisão; número de variáveis utilizadas; número de definições de variáveis; número de nós com definição de variáveis; e número de nós do grafo de programa. Vários modelos de estimativa do número de caminhos não executáveis foram obtidos, notando-se que todas as características influenciam no número de caminhos não executáveis do programa; esses modelos são razoavelmente consistentes [MAL91a, VER92].

Com o propósito de testar a hipótese de Malevris para o "benchmark", foram calculados para cada rotina, o número de predicados dos caminhos executados (executáveis) e o dos que não puderam ser executados (não executáveis). A Tabela 2.1 mostra que foram encontrados entre todos os caminhos requeridos, 317 possuindo número de predicados $q \geq 6$; entre esses, 122 executáveis e 195 não executáveis. O número de caminhos não executáveis encontrados para as rotinas testadas (1432 entre 2509) foi maior que o esperado. Além disso, apenas três das rotinas testadas não apresentaram caminhos não executáveis.

Utilizando-se os dados da Tabela 2.1, foram explorados vários modelos com o objetivo de validar as afirmações de Malevris [MALE90], de que quanto maior o número de predicados de um caminho, maior a probabilidade dele ser não executável. Pôde-se validar essas afirmações para os potenciais-du-caminhos no benchmark considerado, com o número de predicados q do caminho, no intervalo $1 \leq q \leq 9$ [VER92]. O modelo obtido foi

$p = 1.015\ e^{-0.153q}$ com $r^2 = 95.7$, onde o valor de r^2 demonstra o quanto a equação obtida representa bem os pontos da amostra.

Tabela 2.1: Número de Predicados Contidos nos Potenciais-du-caminhos das Rotinas do Benchmark

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	tot
nex	0	19	41	44	90	118	195	247	131	115	138	117	57	31	24	40	14	5	6	1432
exc	19	106	112	103	122	113	122	101	55	47	61	33	25	16	10	17	8	5	2	1077
tot	19	125	153	147	212	231	317	348	186	162	199	150	82	47	34	57	22	10	8	2509

2.2 - Caracterização de Não Executabilidade

A exemplo do trabalho de Hedley e Hennel [HED85], procurou-se agrupar em categorias as principais causas de não executabilidade encontradas nas rotinas do "benchmark". Essas categorias são apresentadas a seguir. Para efeito de simplificação, apresentam-se apenas os trechos ilustrativos das rotinas consideradas. Esses trechos foram instrumentados pela POKE-TOOL, onde a coluna da esquerda indica o número do nó correspondente a cada bloco de comandos do programa.

a. Laços, flags, e variáveis que controlam laços: estão entre as principais causas de não executabilidade. Os casos mais comuns acontecem com laços cujas condições iniciais na entrada do laço são sempre verdadeiras e que serão sempre executados, ou com laços que contêm teste de variáveis que serão satisfeitos apenas uma vez durante a repetição do laço. Outra causa comum de não executabilidade é originada com redefinição e teste de flags dentro ou logo após o laço. Como exemplo, veja a rotina **getone** (Figura 2.1). Note-se que o caminho dado pela sequência (9,10,11,12,2) é não executável, pois no nó 10, *status = ERR* , o predicado *status == OK*, torna-se falso e o próximo nó do caminho é obrigatoriamente o 14. Também o caminho (9,10, ... 16,17) é não executável pois, em 16, o predicado que utiliza o flag *status* é sempre falso, sendo o nó 19 executado.

b. Laços e comandos de seleção com predicados dependentes: Pode haver o caso em que a satisfação de um predicado implique a satisfação de outro, ou ao contrário, a não

satisfação de um predicado implique a satisfação de um outro, gerando assim caminhos não executáveis. Na rotina **command** (Figura 2.2), os dois predicados exigidos não podem ser satisfeitos ao mesmo tempo. Por exemplo, *cmd == UNKNOWN* e *cmd == FI*, torna o caminho (1,3,4) não executável, assim como os caminhos (1,3,5), ... ,(1,3,17).

c. Teste de variáveis imediatamente após a sua definição: Como exemplo desta situação, novamente na rotina **getone** (Figura 2.1), no nó 15, **status = ERR*; esse valor torna falso o predicado do nó 16 e o caminho (14,15,16,17) não executável. Muitas vezes existe a necessidade de verificar se a variável pode ser redefinida ao longo do caminho. O caminho (1,14,16,17,19) é não executável pois **start* não foi redefinido e o predicado, *i <= *start*, é sempre satisfeito neste caso.

d. Dependência de Contexto: São os casos de atribuição a variáveis de valores retornados por funções; estas variáveis podem assumir apenas esses valores. Na rotina **command** os valores possíveis para a variável *cmd* são: *FI,NF,BR,LS,CE,UL,HE,FO,BP,SP, IND,RM,TI,PL, UNKNOWN*. Portanto, o caminho (1,2,3,19) é não executável, pois exige que *cmd* seja diferente dos valores acima.

Existem outras causas de não executabilidade que não se enquadram em nenhuma das categorias citadas acima. Muitas são combinações dos quatro casos apresentados, outras são casos isolados. Deve-se ressaltar que não foi contabilizada a frequência relativa de ocorrência dessas categorias nas rotinas testadas; porém, os casos **a** e **c** foram os mais observados nas rotinas testadas.

Os caminhos mais difíceis de serem analisados foram os caminhos que possuiam um número maior de predicados, principalmente predicados compostos, onde os valores das variáveis podiam assumir várias combinações possíveis. Por exemplo, veja o predicado associado ao nó 14 na rotina **getone**. Para avaliar tais predicados, foi importante determinar fatos associados a cada nó. Se o caminho contém a sequência 1 14 tem-se *num = 0*; este é um fato que pode ser utilizado para avaliar o predicado do nó 14.

Para detectar caminhos não executáveis, também foi útil a determinação de *padrões* de não executabilidade. Na rotina **getone**, qualquer caminho que contiver o subcaminho (ou o padrão) (14,15,16,17) será não executável. Estes recursos, que foram necessários para determinação manual dos caminhos não executáveis, foram incorporados à POKE-TOOL, conforme apresentados na Seção 3.

```
stcode getone(lin,l,num,status)
1   { lstart = *l; *num = 0;
1     if(getnum(lin,l,num,status)==OK)
2     do{ ....
7       if(getnum(lin,l,&pnum,status)==OK)
8         *num = *num + mul * pnum;
9       if(*status==ENDDATA)
10        *status = ERR;
11      }
12    }while(*status==OK);
14    if((*num<0)||(*num >lastln))
15    *status == ERR;
16    if(*status!=ERR)
17        if(*l<=lstart)
18          *status = ENDDATA;
          else
19          *status = OK;
20  } ....
```

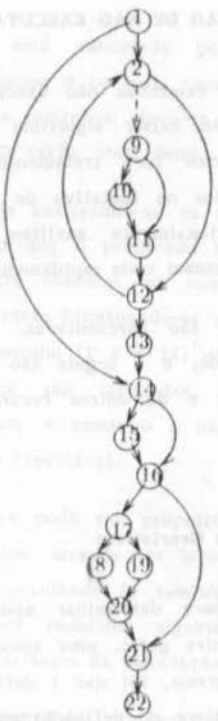


Figura 2.1 Rotina getone

```
void command(buf)
1       { cmd = getcmd(buf);
1       if (cmd != UNKNOWN)
2         val = getval(buf,&argtype);

3       switch(cmd){
4       case FI: ....
5       case NF: ....
6       case BR: ....
          ....
16      case TI:....
17      case PL: ...
18      case UNKNOWN:
        }
19    }
```

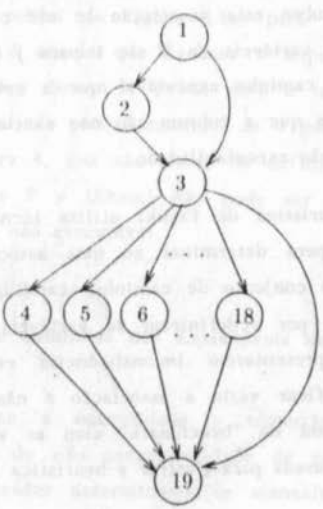


Figura 2.2 Rotina COMMAND

3 - DETERMINAÇÃO DE NÃO EXECUTABILIDADE

A existência de caminhos não executáveis acarreta problemas para as atividades de teste, já que não existe algoritmo para determinar se um dado caminho é ou não executável. Recursos para tratamento de não executabilidade, evitam que esforço e tempo sejam gastos na tentativa de gerar dados de teste para exercitar elementos não executáveis; adicionalmente auxiliam a geração de dados de teste para elementos executáveis e eliminam mais rapidamente esses elementos.

Nesta Seção são apresentadas, através de exemplos, heurísticas para determinar não executabilidade; e a seguir são descritos os principais aspectos de implementação dessas heurísticas e de outros recursos para determinação de não executabilidade, na POKE-TOOL.

3.1 - Aplicação de Heurísticas

Uma heurística para determinar associações não executáveis foi proposta por Frankl [FRA87]. De maneira geral, uma associação [i,j,V] ou [(i,(j,k),V] relaciona um nó i de um grafo de programa, tal que i define as variáveis $v \in V$, a um nó j ou arco (j,k), por um caminho livre de definição com relação a variáveis de V, ou seja, nenhuma das variáveis de V é definida nos nós do caminho de i para j ou para (j,k). Um caminho completo cobre esta associação se ele contiver um subcaminho livre de definição com respeito às variáveis de V de i para j ou para (j,k). A associação é não executável se não existe caminho executável que a cobre. Se uma associação é não executável, todos os caminhos que a cobrem são não executáveis e a associação poderá ser considerada um padrão de não executabilidade.

A heurística de Frankl utiliza técnicas de execução simbólica e análise de fluxo de dados para determinar se uma associação é ou não executável. A idéia básica é eliminar do conjunto de caminhos candidatos a cobrir a associação, aqueles que não são viáveis, ou por redefinirem as variáveis da associação ou por não serem executáveis, ou seja apresentarem inconsistências entre os seus predicados. Se o conjunto de candidatos ficar vazio a associação é não executável. A Figura 3.1a mostra o grafo da rotina **append** do "benchmark" com as variáveis definidas em cada nó do grafo. Esta rotina será usada para ilustrar a heurística de Frankl.

Para a rotina **append** a associação (3,(4,14), (curln, stat, done)) é requerida

pelos critérios Potenciais Usos. Essa associação será executada pelo conjunto de caminhos que saem do nó 1, passam pelo nó 3, executam 0 ou mais caminhos através do laço que começa no nó 4 e vão para o nó 14. Desses caminhos, deve-se eliminar aqueles que não são livres de definição com relação às variáveis **curln, stat, done.**

A segunda parte da heurística deve ser aplicada analisando-se os caminhos através do laço do nó 4. Para sair desse laço é necessário que o predicado P = (idone) && (stat = 2) seja avaliado falso. Por isso é necessário executar um caminho dentro do laço que redefina uma das variáveis de P, **done** ou **stat**. Eliminando-se os caminhos que não redefinem essas variáveis, obtém-se um único caminho (1 3 4 14) equivalente a não executar o laço. Técnicas de avaliação simbólica são utilizadas para avaliar o predicado P; ao executar (1 3 4), P é avaliado true, o caminho é não executável. O conjunto de candidatos ficou vazio e a associação é não executável.

Uma otimização (extensão 1) para essa heurística pode ser proposta, observando-se o exemplo dado. Nota-se que a análise dos caminhos através do laço do nó 4 não precisaria ser realizada, pois todas as variáveis do predicado P, também são variáveis da associação analisada e, portanto, seria impossível redefinir alguma das variáveis de P por caminhos livres de definição com relação às variáveis da associação.

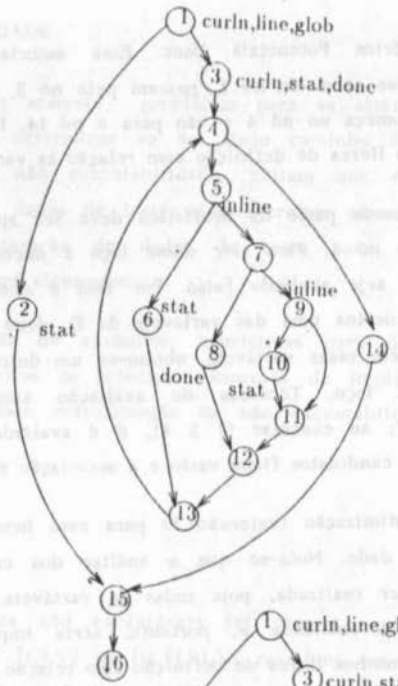
Existem casos nos quais a determinação de caminhos não executáveis pode ser realizada mesmo sem a avaliação de predicados, podendo-se propor uma segunda extensão (extensão2) para a heurística de Frankl. Seja a rotina da Figura 3.1b; para cobrir a associação [8,(4,14),(stat)], um caminho candidato é aquele que executa a sequência (8 10 11 12 13), 0 ou mais caminhos através do laço do nó 4 e vai para o nó 14. O predicado do nó 4, P = (idone) não precisa ser avaliado pois o caminho de 4 para 8 não redefine **done**, e novamente, os caminhos de 8 para 4, que são livres de definição com respeito a **stat**, não redefinem **done**. Portanto, se P ≠ (idone) não pode ser avaliado true, então o nó 14 não é alcançado e a associação é não executável.

3.2 - Incorporação de Recursos para Tratamento de Caminhos não Executáveis na POKE-TOOL.

Durante a condução do "benchmark", verificou-se a necessidade e importância de incorporar à POKE-TOOL recursos para tratamento de não executabilidade de caminhos. Muitos dos recursos implementados foram estabelecidos determinando-se manualmente a executabilidade dos caminhos e associações e estudando-se as principais causas de não executabilidade encontradas, entre essas os laços e flags.

```
stcode append (line,glob)
int line, glob;
1 {
1  char inline[50];
1  stcode stat;
1  int done;
1  if(glob)
2  stat = 1;
3 else
3 {
3  curln = line;
3  stat  = 2;
3  done = 0;
4  while((!done)&&(stat==2))
5  {
5   if(!getline(inline,stdin,50))
6     stat = 0;
7    else
7    {
7     if((inline[0]==',')
        &&  (inline[1]==13))
8       done = 1;
9     else
        if(puttxt(inline)==1)
10       stat  = 1;
11
12  }
13 }
14
15 }
16 return stat;
16 }
```

a)



b)

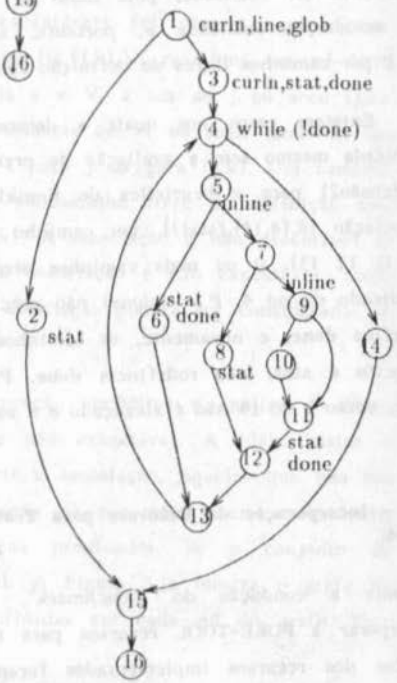


Figura 3.1: Ilustração de Heurísticas

A POKE-TOOL é uma ferramenta que apóia a utilização dos Critérios Potenciais Usos para o teste de unidades, procedimentos de uma linguagem de programação. A POKE-TOOL tem como entrada o programa a ser testado, a seleção do critério a ser utilizado e um conjunto de casos de teste. Na hipótese deste conjunto ser vazio, a POKE-TOOL fornecerá um conjunto de caminhos/associações necessários para satisfazer o critério selecionado, orientando desta forma a seleção de casos de teste. Se o conjunto não for vazio, será produzida uma relação de caminhos requeridos pelo critério mas ainda não executados. Muitos desses caminhos ainda não executados podem ser não executáveis. Para tratar esses caminhos foram incorporados os seguintes recursos:

- determinar associações não executáveis através de heurísticas; aplicação da heurística proposta por Frankl e extensões.
- determinar caminhos não executáveis obtendo-se inconsistências de predicados nesses caminhos, ou um padrão não executável dado.
- eliminar automaticamente caminhos e associações não executáveis.
- determinar os predicados associados aos nós de um caminho dado.
- gerar os possíveis caminhos que cobrem uma associação.
- executar simbolicamente os comandos para auxiliar a avaliação de predicados.
- avaliar predicados associados aos nós, a partir da execução simbólica e de fatos e informações recebidas do usuário.

Resumidamente, dada uma potencial-associação são determinados os possíveis candidatos a cobrirem a associação dada. Para determinar se a associação é não executável, é aplicada a heurística proposta por Frankl e extensões, onde são analisados os predicados de cada laço, e verificada a possibilidade de eliminar os candidatos, utilizando-se avaliação simbólica e padrões de não executabilidade. Se todos os conjuntos puderem ser eliminados a associação será não executável e deverá ser eliminada do conjunto de associações requeridas, assim como os caminhos que a cobrem, constituindo-se um novo padrão de não executabilidade. A executabilidade de caminhos é determinada verificando-se inconsistências na avaliação dos predicados associados ao caminho ou verificando a existência de um padrão. Os padrões, que incluem associações e caminhos não executáveis, também podem ser fornecidos pelo usuário, e assim serão eliminados os caminhos e associações correspondentes. A avaliação de predicados de um nó é realizada baseando-se em todas as informações que o usuário possa oferecer e num conjunto de valores para as variáveis do programa obtidos pela execução simbólica.

4 - CONCLUSÕES

Erros estão presentes na maioria dos programas e está clara a necessidade de automatizar técnicas de teste para obtenção de produtos confiáveis e de baixo custo. Neste trabalho, procurou-se mostrar os principais problemas introduzidos por caminhos não executáveis na automatização das atividades de teste. Na prática, critérios de teste estruturais exigem a execução de elementos não executáveis e não se pode gerar dados de teste para satisfazer esses critérios.

A causa mais comum de não executabilidade dos caminhos não executáveis encontrados nas rotinas do "benchmark" foi o uso de flags para os laços. Muitas das causas apresentadas poderiam ser eliminadas reescrevendo-se o código. Baseando-se nas categorias descritas na Seção 2.2, podem ser propostas técnicas para se obter programas com um número reduzido de caminhos não executáveis; por exemplo, a utilização de um número menor de flags. Estudos estão sendo conduzidos para também propor novas heurísticas para determinar não executabilidade.

A hipótese de Malevris, sobre a influência do número de predicados q de um caminho na sua executabilidade, foi comprovada. Para as rotinas do "benchmark" considerado, quanto maior o número de predicados menor a probabilidade do caminho ser executável. Isso pode ser utilizado como estratégia para gerar dados de teste. Por exemplo, selecionar entre dois caminhos para cobrir uma associação aquele que contiver o menor número de predicados, pois a probabilidade deste ser executável é maior.

Os recursos para tratamento de não executabilidade que foram incorporados à ferramenta POKE-TOOL auxiliam a geração de dados de teste para elementos requeridos pelos critérios Potenciais Usos, possibilitam a eliminação de padrões não executáveis automaticamente, implementam heurísticas e verificam inconsistências de predicados; isso permite evitar que esforço e tempo sejam gastos tentando cobrir elementos não executáveis e reduz os custos das atividades de teste.

A utilização de padrões não executáveis é muito importante pois em algumas rotinas um dado padrão está contido num número muito grande de caminhos. As extensões propostas para a heurística de Frankl permitem simplificações para implementação. A extensão 1 torna desnecessário entrar em detalhes sobre um laço que está sendo considerado. A extensão 2 permite determinar não executabilidade sem necessidade de avaliar o predicado do laço.

Para implementar os recursos propostos foram encontradas inúmeras restrições

[VER92], tais como: tratamento de ponteiros, chamadas de procedimentos e variáveis globais. Essas restrições estão relacionadas à execução simbólica dos comandos do programa e à avalição dos predicados. Essas limitações puderam ser bastante reduzidas através da interação com o usuário; este poderá fornecer padrões de não executabilidade, fatos que auxiliarão a avaliar os predicados, etc. Essas informações podem ser úteis no processo de depuração e manutenção. Pretende-se numa nova etapa, implementar técnicas de avaliação simbólica mais poderosas e também incorporar à POKE-TOOL recursos para geração de dados de teste, depuração e manutenção de programas.

REFERÊNCIAS

[CHA91] M.L.Chaim, "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, Brasil, Abril 1991.

[FRA86] F.G.Frankl, E.J.Weyuker, "Data Flow Testing in the Presence of Unexecutable Paths", in Proc. Workshop on Software Testing, Banff, Canada, pp 4-13, Jul. 1986.

[FRA87] F.G.Frankl, "The use of Data Flow Information for the Selection and Evaluation of Software Test Data," Ph.D Dissertation, New York, Oct. 1987.

[HED85] D.Hedley e M.A.Hennell, "The Causes and Effects of Infeasible Paths in Computer Programs", Proc. 8th. ICSE London, UK (1985) pp 259-266.

[KER81] B.W.Kernighan e P.J.Plauger, Software Tools in Pascal. Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.

[MAY90] A.Mayhauser, Software Engineering - Methods and Management, Academic Press Inc, USA, 1990.

[MAL88] J.C.Maldonado, M.LChaim, M.Jino, "Seleção de Casos de Testes Baseada nos Critérios Potenciais Usos", II Simpósio Brasileiro de Engenharia de Sotware, Canela, RS, Brasil, Out. 1988, pp. 24-35.

[MAL91a] J. C.Maldonado, "Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software", Tese de Doutorado, DCA/FEE/UNICAMP Campinas-SP, Brasil, Julho 1991.

[MAL91b] J.C.Maldonado, M.L.Chaim, S.R.Vergílio, M.Jino, "Critérios Potenciais Usos: Uma Contribição para a Atividade de Garantia de Qualidade de Software", in Proc. Workshop em Avaliação de Qualidade de Software, COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Maio 1991.

[MALE90] N.Malevris, D.F.Yates, A.Veevers, "Predictive Metric for Likely Feasibility of Program Paths". Information and Software Technology, Vol.32, No.2, March 1990.

[MYE79] G.J.Myers, "The Art of Software Testing", Wiley, 1979.

[RAP82] S.Rapps, E.J.Weyuker, "Data Flow Analysis Techniques for Test Data Selection", in Proc. Int. Conf. Software Eng., Tokio, Japão,Set 1982.

[RAP85] S.Rapps, E.J.Weyuker, "Selecting Software Test Data Using Data Flow Information," IEEE, Trans. Sofware Eng., Vol. SE - 11,pp. 367-375, Apr 1985.

[VER92] S.R.Vergilio, "Caminhos Não Executáveis: Caracterização, Previsão e Determinação para Suporte ao Teste de Programas", Tese de Mestrado DCA/FEE/UNICAMP, Campinas -SP ,Janeiro 1992.

[WEY84] E.J.Weyuker, "The Complexity of Data Flow Criteria for Test Data Selection," Information Processing Letters, Vol. 19, N. 12, pp 121-128, Feb. 1990.

[WEY90] E.J.Weyuker, "The Cost of Data Flow Testing: An Empirical Study", IEEE Trans. Soft. Eng., Vol. SE-16, No 2, Feb. 1990, pp.121-128.