

CAMINHOS NÃO EXECUTÁVEIS: CARACTERIZAÇÃO, PREVISÃO E
DETERMINAÇÃO PARA SUPORTE AO TESTE DE PROGRAMAS.

SILVIA REGINA VERGILIO ^{m 86}

Orientador: Prof. Dr. MARIO JINO

Co-orientador: José Carlos Maldonado

Dissertação apresentada à Faculdade de Engenharia Elétrica
da UNICAMP, como requisito parcial para obtenção do Título
de Mestre em Engenharia Elétrica.

Campinas, Janeiro de 1992

Este exemplar corresponde à defesa final da tese
defendida por Silvia Regina Vergilio

e foi avaliada pela Comissão
Julgadora em 30 / 01 / 92

Mario Jino
Orientador

Aos meus Pais

Agradecimentos

Gostaria de agradecer ao meu orientador, Prof. Dr. Mario Jino, pela orientação, pela amizade e pelo meu ingresso no grupo de Teste de Software.

Ao Prof. Dr. José Carlos Maldonado pela orientação, motivação, sugestões e críticas. Sem dúvida, suas observações, sua amizade e paciência, e seu otimismo foram fundamentais para a realização deste trabalho.

A todo grupo de Teste de Software: Marcos L. Chaim, Rubens Pontes Fonseca e Plínio de Sá Leitão, pelo companheirismo demonstrado sempre e principalmente durante os cafezinhos.

A todos os amigos e colegas da Faculdade de Engenharia Elétrica e do Instituto de Matemática, Estatística e Computação da Unicamp, pela amizade, pelo excelente ambiente de trabalho e pelo voley da sexta.

Ao Heraldo Maciel França Madeira, pelo carinho, apoio e sugestões dados durante o desenvolvimento deste trabalho.

Aos amigos da rua Luverci Pereira de Souza: Carol, Cecil, Chris, Elisa e Klaus, Gilberto, Ivonne, Inés, Marcos, Raquel e Valéria pelo carinho e alegrias do dia a dia.

Aos meus pais Wilson e Maria Elvira, e a toda a minha família que sempre deram a maior força.

A CAPES pelo apoio financeiro.

RESUMO

Neste trabalho são discutidos os principais problemas introduzidos por caminhos não executáveis nas atividades de teste de programas, já que é indecidível se um caminho é ou não executável. O trabalho enfoca três aspectos principais: caracterização, previsão e determinação de caminhos não executáveis. Os estudos foram realizados baseando-se em trabalhos existentes na literatura e em resultados obtidos durante a condução de um "benchmark". Para isto, utilizou-se uma ferramenta de testes, denominada POKE-TOOL, que apóia a aplicação dos critérios Potenciais-Usos. São apresentados: as principais causas de não executabilidade encontradas nas rotinas do "benchmark"; modelos para avaliar a influência de várias características de programas, no número de caminhos não executáveis e modelos para avaliar a relação entre o número de predicados do caminho e sua executabilidade. A condução do "benchmark" também ressaltou a importância da aplicação das heurísticas propostas por Frankl [FRA87] para identificação de elementos não executáveis; além disto, levou a proposição de extensões para esta heurística e viabilizou a identificação de facilidades que foram incorporadas na POKE-TOOL, para tratamento de tais elementos. Adicionalmente, são apresentados: os principais aspectos de implementação das heurísticas e facilidades propostas, um exemplo de utilização das rotinas implementadas e uma avaliação preliminar do desempenho das mesmas.

ABSTRACT

This work discusses the main problems introduced by infeasible paths in the activities of program testing, since it is undecidable whether a path is or is not feasible. The work focuses on three major aspects: classification, estimation and determination of infeasible paths. The studies were accomplished based on results reported in the literature and on results taken from the application of a benchmark. To conduct the benchmark, the testing tool used was the POKE-TOOL, a tool which supports the Potential-Uses criteria. The main causes for non-feasibility of paths in the benchmark's routines are presented. Models which assess the influence of several characteristics of programs on the number of infeasible paths and models to assess the relation between the number of predicates in paths and their feasibility are also presented. The benchmark pointed out the relevance of the Frankl's heuristic application [FRA87] for identification of infeasible paths; more over, has made possible the proposition of extension to this heuristic and of facilities to deal with such paths, which were incorporated into POKE-TOOL. The heuristics and main aspects of implementation of the proposed facilities are presented; an example of utilization and a preliminary assessment of the effectiveness of the implementation are also shown.

CONTEÚDO

1 INTRODUÇÃO	
1.1 Apresentação	1
1.2 Motivação	7
1.3 Objetivos e Organização da Tese	8
2 REVISÃO BIBLIOGRÁFICA	
2.1 Conceitos Gerais - Terminologia	10
2.2 Caracterização e Previsão de Caminhos não Executáveis	15
2.3 Determinação de Executabilidade	19
2.4 Critérios Potenciais Usos	22
2.5 Considerações Finais	30
3 RESULTADOS DA APLICAÇÃO DE UM "BENCHMARK"	
3.1 Realização e Coleta de Resultados	32
3.2 Caracterização dos Caminhos não Executáveis	37
3.3 Características do Programa X Não Executabilidade	44
3.4 Número de Predicados X Executabilidade de Caminhos	47
3.5 Considerações Finais	51
4 DETERMINAÇÃO DE EXECUTABILIDADE NAS ATIVIDADES DE TESTE	
4.1 Aspectos Teóricos da Implementação	55
4.1.1 Aplicação de Heurísticas	55
4.1.2 Padrões de Não Executabilidade	63
4.1.3 Execução Simbólica	64
4.1.4 Avaliação de Predicados	69
4.2 Aspectos da Arquitetura e da Implementação da POKE-TOOL	72

4.3 Incorporação de Facilidades à POKE-TOOL para Tratamento de Não Executabilidade	77
4.4 Considerações Finais	82
5 ASPECTOS DE IMPLEMENTAÇÃO	
5.1 O Módulo Pokernel da POKE-TOOL	89
5.2 O Módulo de Apoio à Geração de Casos de Teste	95
5.3 Exemplo de Utilização das Rotinas Implementadas	103
5.4 Considerações Finais	117
6 CONCLUSÕES E TRABALHOS FUTUROS	
6.1 Conclusões	119
6.2 Trabalhos Futuros	122
 REFERÊNCIAS	123
 A PRINCIPAIS MODELOS OBTIDOS DA ANÁLISE DE EXECUTABILIDADE DE CAMINHOS NAS ROTINAS DO "BENCHMARK" APLICADO	
A.1 Potenciais-dcaminhos X Número de Predicados	128
A.2 Melhores Modelos para Avaliar a Influência de Várias Características do Programa no Número de Caminhos não Executáveis	132
A.3 Melhores Modelos Obtidos para Avaliar a Influência do Número de Predicados de um Caminho na sua Executabilidade	139
A.3.1 Melhores Modelos para o Conjunto TOTAL	139
A.3.2 Melhores Modelos para o Conjunto TOTAL-R	144
 B EXEMPLOS DE UTILIZAÇÃO DAS FACILIDADES IMPLEMENTADAS	
B.1 Rotina APPEND	152
B.2 Rotina GETFNS	161

LISTA DE FIGURAS

2.1 Exemplo da Heurística Proposta por Frankl	21
2.2 Ordem Parcial entre a Família de Critérios Potenciais Usos e a Família de Critérios de Fluxo de Dados	28
2.3 Ordem Parcial entre a Família de Critérios Potenciais Usos e a Família de Critérios de Fluxo de Dados na Presença de Caminhos Não Executáveis	29
3.1 Caminhos Não executáveis Gerados por laços que são sempre executados	38
3.2 Rotina GETONE	39
3.3 Rotina GETFNS	40
3.4 Rotina COMMAND	41
3.5 Rotina TRANSLIT	43
4.1 Exemplo de Utilização da Heurística de Frankl	58
4.2 Grafos(i) Gerados pelo Programa da Figura 4.1	59
4.3 Exemplo para a Extensão 2 da Heurística de Frankl	61
4.4 Contra - Exemplo para a Extensão 2 da Heurística de Frankl	62
4.5 Problema com Arrays na Execução Simbólica	67
4.6 Problema com Chamada de Procedimentos e Funções na Execução Simbólica	68
4.7 Exemplo de Introdução de Fatos pelo Usuário na Avaliação de Predicados	71
4.8 Arquitetura da Ferramenta POKE-TOOL	76
4.9 Introdução de Novas Funções na POKE-TOOL	80
4.10 Detalhamento da Função Apoio à Geração de Casos de Teste	81
5.1 Projeto da POKE-TOOL	87
5.2 Projeto Modificado da POKE-TOOL	88
5.3 Exemplo de Grafo-def/pred para o Programa da Figura 4.1 do Capítulo 4	92
5.4 Representação de um Comando para Execução Simbólica	Instrumentado
	94

LISTA DE TABELAS

2.1 Amostra Utilizada por Malevris	18
3.1 Variáveis de Controle Relativas ao Benchmark	34
3.2 Variáveis Respostas: Informação Dinâmica Relativa ao Benchmark	35
3.3 Número de Predicados Contidos nos Potenciais-du-Caminhos das Rotinas do Benchmark	36
3.4 Número de Predicados Contidos nos Potenciais-du-Caminhos das Rotinas do Benchmark Excluindo-se as Unidades Recursivas	36
3.5 Melhores Modelos Obtidos para os Critérios Potenciais Usos	46
3.6 Número de Predicados Contidos nos Potenciais-du-Caminhos das Rotinas do Benchmark com $q \geq 12$	50
3.7 Número de Predicados Contidos nos Potenciais-du-Caminhos das Rotinas do Benchmark com $q \geq 12$, Excluindo-se as Rotinas Recursivas	50
5.1 Avaliação das Rotinas Implementadas	118

CAPÍTULO 1

INTRODUÇÃO

1.1 Apresentação

O desenvolvimento de software está sujeito a uma série de tipos de erros. Esses erros, contrariamente ao processo de fabricação, que envolve produtos físicos, são erros humanos, originados na maioria das vezes por falhas de transformação e comunicação nas diversas fases de desenvolvimento do software. Por isso têm sido propostas técnicas de engenharia de software, com o objetivo principal de produzir software de alta qualidade [PRE87]. Dentro da engenharia de software o teste é uma atividade fundamental entre as atividades de garantia de qualidade, ou seja, é uma parte essencial no desenvolvimento de software [HOW87].

As atividades de teste devem fazer parte do planejamento global do sistema; a falta de tempo e de recursos humanos capacitados e a não disponibilidade de ferramentas adequadas são os principais problemas enfrentados pelas equipes de teste. Além disso, as atividades de teste consomem da ordem de 50% do tempo e custo de desenvolvimento de software [MYE79]. A atividade de teste deve envolver: planejamento de testes, projeto de casos de teste, execução e avaliação dos resultados dos testes.

Segundo Myers [MYE79], teste é o processo de execução de um programa com o objetivo de encontrar erros. Um bom caso de teste é aquele que tem alta probabilidade de encontrar um erro ainda não descoberto. O teste é dito bem sucedido se encontrou um erro ainda não detectado. Como objetivo secundário, o teste de software demonstra que

as funções trabalham aparentemente como o desejado e que os requisitos de desempenho foram cumpridos. Além do mais, testes não podem mostrar a ausência de erros mas somente indicar a presença desses.

Idealmente, o programa deveria ser exercitado para todos os valores de entrada possíveis, mas o teste exaustivo é impraticável, devido a restrições de tempo e de custo. Por isso, muitas técnicas surgiram nas últimas décadas para projetar casos de teste efetivos, visando a oferecer uma maneira sistemática para determinar qual o subconjunto de todos os possíveis casos de teste que pode detectar a maioria dos erros, com o mínimo de tempo e esforço. Pressman [PRE87] define entre essas técnicas duas que podem ser consideradas básicas: teste estrutural (ou teste de caixa branca), no qual os casos de teste são produzidos a partir da estrutura lógica dos programas e teste funcional (ou teste de caixa preta), onde os casos de teste são derivados a partir dos requisitos funcionais, sem conhecimento da estrutura interna do programa. Myers [MYE79] apresenta uma boa estratégia de aplicação dessas duas técnicas e diz que elas devem ser consideradas complementares, pois cobrem diferentes classes de erros.

A técnica estrutural está baseada no conhecimento da estrutura interna e da implementação do programa. O objetivo é caracterizar um conjunto de elementos (que podem ser arcos, comandos, caminhos) que devem ser exercitados pelo conjunto de casos de teste. Recentemente, foram introduzidos critérios baseados em análise de fluxo de dados para selecionar casos de teste [HEC77, HER76, NTA84, LAS83, RAP82, URA88, MAL88a]. A idéia básica é executar os caminhos do ponto onde uma variável do programa foi definida até o ponto onde ela foi usada. A análise de fluxo de dados estuda as maneiras pelas quais os valores atribuídos a cada variável podem afetar a execução do programa e foi utilizada originalmente na otimização de código por compiladores.

Comparações entre os diversos critérios estruturais de teste, têm sido conduzidas, baseadas principalmente no estudo da complexidade dos critérios - número de casos de teste exigidos pelo critério no pior

caso; e no estudo da ordem parcial entre esses critérios, estabelecida por uma relação de inclusão [CLA86, RAP85, WEY84, NTA88, MAL89b, MAL91a], apresentada na seção 2.4 do Capítulo 2.

A ordem parcial entre esses critérios engloba algumas propriedades mínimas que devem ser satisfeitas para o estabelecimento de novos critérios. Por exemplo, incluir o critério todos os ramos e, do ponto de vista de fluxo de dados, incluir o uso de todo resultado computacional [CLA86, MAL91a].

Os elementos exigidos pelos critérios estruturais podem conter e, em geral, contêm elementos não executáveis. Isto porque os critérios fazem a escolha dos caminhos que devem ser executados, baseada apenas na sintaxe do programa. Um caminho é dito executável (ou factível) 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 (ou não factível) [FRA87]. A presença de caminhos não executáveis perturba consideravelmente a ordem parcial entre os critérios, isto é, algumas propriedades passam a não ser válidas. Por exemplo, os 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.

Nota-se, portanto, que existem duas questões relacionadas à fase de teste: como selecionar os dados de teste e como saber se um programa foi testado o suficiente. À primeira questão está associado um método M para escolher casos de teste e à segunda um critério C de adequação de um dado conjunto de casos de teste; este é um predicado que deve ser satisfeito para considerarmos um programa testado o suficiente segundo o critério. Frankl [FRA86] observa que existe uma correspondência entre um

método de seleção de dados de teste e um critério de adequação. Dado um critério de adequação C , existe um método M_C que diz: "selecione um conjunto de teste que satisfaz C " e dado um método M , existe um critério C_M que diz: um conjunto de teste é adequado se ele foi gerado pelo método M ". Nessa tese o termo critério será usado com ambos os sentidos.

Segundo Weyuker [WEY86], um bom critério de teste deve satisfazer a propriedade da aplicabilidade. Um critério C satisfaz a propriedade da aplicabilidade se, para todo programa P , existe um conjunto de casos de testes T , C -adequado para P . Para que um conjunto T de casos de teste seja C -adequado para um programa P , o conjunto de caminhos Π executados por T deve cobrir cada elemento exigido pelo critério C . Entretanto, nenhum dos critérios estruturais, inclusive os critérios baseados em fluxo de dados, satisfaz essa propriedade; mais ainda, é indecidível se para um dado programa P existe um conjunto de casos de teste T que é C -adequado para P , devido ao aspecto de executabilidade dos caminhos [FRA86, FRA87].

Para viabilizar a automatização desses critérios, foi derivado para cada critério C um novo critério C' , denominado critério executável. Esses critérios executáveis satisfazem a propriedade da aplicabilidade. Ou seja, para qualquer programa P e qualquer critério executável, existe um conjunto de casos de teste T que satisfaz o critério, pois esse exige apenas a execução de caminhos executáveis. Entretanto, não é possível dizer em geral se um dado conjunto de casos de testes T satisfaz um critério executável, para um dado programa P . Para sabermos se um conjunto de casos de teste T satisfaz os critérios executáveis é necessário sabermos quais caminhos do programa são executáveis. Embora na maioria das vezes seja fácil para o ser humano identificar se um caminho é ou não executável, não existe algoritmo que para um dado caminho decida sua executabilidade [FRA87]; esse é um problema equivalente ao problema da parada de programas [MAY90].

A existência de caminhos não executáveis impede que se gerem dados de teste para execução dos 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, e os critérios executáveis é que na verdade são utilizados. Além disso, os custos aumentam consideravelmente devido à não executabilidade de caminhos [MAY90]; no caso de teste de ramos, os fatores de tempo e esforço são fortemente influenciados pelo número de caminhos não executáveis [MALE90].

Segundo Malevris a existência de caminhos não executáveis foi documentada por volta de 1972, e desde então vários problemas introduzidos por esses caminhos foram reconhecidos, particularmente os problemas relativos aos critérios de teste de programas e à automatização desses critérios. Surpreendentemente, poucos autores têm-se dedicado aos problemas relativos a caminhos não executáveis [MALE90].

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 as heurísticas propostas por Frankl [FRA87] para determinar associações não executáveis.

Maldonado, Chaim e Jino definiram os critérios Potenciais Usos e os correspondentes critérios Potenciais Usos Executáveis [MAL88a, MAL89b]. Esses critérios exigem a execução de caminhos livres de definição, a partir da definição de uma determinada variável do programa, independentemente da ocorrência de um uso desta variável nesses caminhos. Neste caso, pode-se verificar se o valor de x não foi alterado (possivelmente por efeitos colaterais) ao longo dos caminhos executados. Adicionalmente, podem auxiliar na identificação de falhas causadas por dependências de fluxo de dados ausentes, originados por usos ausentes de variáveis.

Os critérios Potenciais Usos exigem menos informação sobre o fluxo de dados do programa do que os demais critérios baseados em análise de fluxo de dados, pois não necessitam saber como as variáveis definidas estão sendo utilizadas, são os únicos critérios baseados em análise de fluxo de dados que, mesmo na presença de caminhos não executáveis, satisfazem os requisitos exigidos para o estabelecimento de testes estruturais [MAL89b,MAL91a].

Para dar suporte à utilização dos critérios Potenciais Usos foi implementada uma ferramenta de teste, denominada POKE-TOOL [CHA91d]. Esta ferramenta está operacional para programas escritos na linguagem C e foi desenvolvida para ambientes do tipo PC. Tem o objetivo de determinar caminhos e associações requeridas e avaliar um conjunto de casos de teste com relação a esses critérios.

Para verificar o comportamento e aplicação dos critérios Potenciais Usos em programas reais, e estudar a ocorrência e influência de caminhos não executáveis, 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].

Para a condução do "benchmark" foi especificado, para cada rotina, um conjunto de casos de teste, baseando-se apenas nos aspectos funcionais de cada uma das rotinas. Esses casos de teste foram executados e avaliados pela POKE-TOOL que produziu uma lista de caminhos e associações exigidas pelos critérios Potenciais Usos e que não foram executados pelo conjunto de casos de teste inicial. Para gerar novos casos de teste para os elementos restantes, os caminhos e associações não executáveis e a causa da não executabilidade foram detectados manualmente.

Apesar das rotinas do "benchmark" serem programas bem formulados, o número de caminhos e associações não executáveis encontrado foi maior que o esperado. E além disso, identificar manualmente a executabilidade

dos caminhos e associações foi uma tarefa tediosa que consumiu muito esforço e tempo.

1.2 Motivação

A partir da apresentação dada e da experiência obtida da condução do "benchmark" podem ser destacados alguns fatores que serviram como motivação para essa tese.

- 1) erros estão presentes na maioria dos programas, e está clara a necessidade de automatização de técnicas de teste para obtenção de produtos confiáveis e de baixo custo.
- 2) caminhos e associações não executáveis foram encontrados em grande número na maioria das rotinas testadas (1432, 55.9% do número total de caminhos).
- 3)a existência de caminhos não executáveis perturba consideravelmente a ordem parcial entre os critérios baseados em fluxo de dados.
- 4) na prática, os critérios não podem ser satisfeitos, isto é, nenhum satisfaz a propriedade da aplicabilidade, e os critérios executáveis é que são realmente utilizados porque satisfazem essa propriedade.
- 5) identificar manualmente não executabilidade é uma tarefa que consome bastante esforço e tempo, aumentando consideravelmente o custo das atividades de teste.

Portanto, oferecer facilidades para manipulação e identificação de caminhos não executáveis é uma atividade essencial e de grande contribuição para as atividades de teste e validação de software.

1.3 Objetivos e Organização da Tese

Conforme caracterizado na seção anterior, o objetivo desta tese é estudar o problema de não executabilidade e incorporar facilidades para tratamento de caminhos não executáveis à ferramenta POKE-TOOL. Para isso realizou-se um estudo sobre os principais trabalhos existentes na literatura. O trabalho foi dividido em três partes. A primeira parte diz respeito à caracterização de não executabilidade; visa a apresentar as principais causas de não executabilidade de caminhos entre as rotinas do "benchmark".

A segunda parte consiste em analisar os dados obtidos na aplicação do benchmark, para estudar o relacionamento, como estudado por Malevris [MALE90], entre o número de predicados de um caminho e sua executabilidade. E o relacionamento entre o número de caminhos não executáveis e várias características do programa, tais como número de variáveis, número de definições, número de comandos condicionais, etc, objetivando estabelecer uma maneira de prever não executabilidade.

A terceira parte trata como incorporar facilidades à POKE-TOOL, para determinar e eliminar elementos não executáveis; essas facilidades foram estabelecidas através de heurísticas propostas por Frankl [FRA87], e através de extensões obtidas durante a condução do "benchmark" e de interações com o usuário testador.

Neste Capítulo foram introduzidos o problema de caminhos não executáveis nas atividades de teste, os objetivos e as motivações deste trabalho.

No Capítulo 2 são apresentados os principais trabalhos existentes na literatura com respeito à não executabilidade, e também os critérios Potenciais Usos implementados pela POKE-TOOL, juntamente com a terminologia utilizada; consiste essencialmente a revisão bibliográfica.

No Capítulo 3 são apresentados os resultados obtidos da aplicação do "benchmark" com relação à não executabilidade, mais particularmente com relação à caracterização e previsão de caminhos não executáveis.

No Capítulo 4, após uma descrição da ferramenta POKE-TOOL, é apresentada uma descrição sobre as facilidades que foram incorporadas, e para tratamento (determinação e eliminação) de não executabilidade, e são feitas algumas considerações com respeito a essas facilidades.

No Capítulo 5 são apresentados os aspectos de implementação dessas facilidades e um exemplo de utilização, bem como os resultados preliminares obtidos.

No Capítulo 6 são apresentadas as conclusões e propostas de trabalhos futuros nessa linha de pesquisa.

O trabalho contém dois apêndices. No Apêndice A estão organizados informações relevantes dos modelos de previsão de caminhos não executáveis apresentados no Capítulo 3. No Apêndice B estão organizados dois exemplos extraídos do benchmark com o objetivo de ilustrar as facilidades incorporadas à Poketool para determinar caminhos não executáveis.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados a terminologia e os conceitos básicos pertinentes às atividades de teste, particularmente ao teste estrutural, e são introduzidas notações que serão utilizadas neste trabalho.

Também é apresentada uma revisão bibliográfica dos principais trabalhos existentes na literatura que abordam o problema de não executabilidade. Adicionalmente, são apresentados os critérios Potenciais Usos cuja aplicação é apoiada pela ferramenta de teste POKE-TOOL [MAL89a CHA91d].

2.1 Conceitos Gerais - Terminologia

Seja o grafo de fluxo de controle dado por $G(N, A, s)$, onde N é o conjunto de nós, A o conjunto de arcos e s o nó de entrada. Considera-se que o grafo G contém um único nó de entrada s e um único nó de saída e . Define-se um caminho como sendo uma sequência finita de nós (n_1, \dots, n_k) $k \geq 2$, $i = 1, 2, \dots, k-1$; tal que (n_i, n_{i+1}) é um arco do grafo G ; diz-se que: um caminho é completo se o primeiro nó do caminho é o nó de entrada e o último nó é o nó de saída do grafo G ; simples se todos os nós, exceto possivelmente o primeiro e o último, forem distintos; e livre de laços se todos os nós são distintos.

Uma ocorrência de variável num programa pode ser de três tipos: definição, uso, indefinição.

Uma definição de variável ocorre se um valor é armazenado na posição de memória. No caso de um programa, a definição ocorre se a variável é referenciada do lado esquerdo de um comando de atribuição, ou em um comando de entrada, ou em chamadas de procedimento como parâmetro de saída (nesse caso, se a variável foi passada por referência ou por nome).

Um uso de uma variável ocorre quando ela é referenciada sem ser definida e pode ser de dois tipos: c-uso("computation-use") e p-uso("predicate-use"). O primeiro afeta diretamente uma computação sendo realizada ou permite que o resultado de uma definição seja observado. O segundo afeta diretamente o fluxo de controle do programa.

Uma variável está indefinida quando ou seu valor se torna inacessível, ou sua localização deixa de estar definida na memória.

Seja x uma variável de um programa. Um caminho (i, n_1, \dots, n_m, j) , $m \geq 0$ é um caminho livre de definição com respeito a (c.r.a.) x do nó i para o nó j ou do nó i para o arco (n_m, j) , se nenhuma definição de x ocorre nos nós n_1, \dots, n_m . Com essa definição o arco (i, j) é um caminho livre de definição c.r.a x do nó i para o arco (i, j) .

Um nó i do grafo G , é um nó cabeça de laço se, realizando-se uma busca em profundidade em G a partir do nó de entrada s , existe um caminho $\pi = (s, \dots, i, \dots, j, i)$, o arco (j, i) é o arco de volta do laço.

Define-se: $\text{defg}(i) = \{\text{variáveis } v / v \text{ é definida no nó } i\}$; $\text{pdcu}(x, i) = \{\text{nós } j / \text{existe um caminho livre de definição c.r.a } x \text{ do nó } i \text{ para o nó } j\}$; $\text{pdpu}(x, i) = \{\text{arcos } (j, k) : \text{existe um caminho livre de definição c.r.a } x \text{ de } i \text{ para o arco } (j, k)\}$; potencial du-caminho c.r.a x como um caminho livre de definição $(n_1, n_2, \dots, n_j, n_k)$ c.r.a x do nó n_1

para o nó n_k e para o arco (n_j, n_k) , onde o caminho (n_1, n_2, \dots, n_j) é um caminho livre de laço e no nó n_i ocorre uma definição de x ; associação potencial-definição-c-uso como a tripla $[i, j, x]$ onde $x \in \text{defg}(i)$ e $j \in \text{pdcu}(x, i)$; associação-potencial-definição-p-uso como a tripla $[i, (j, k), x]$ onde $x \in \text{defg}(i)$ e $(j, k) \in \text{pdpu}(x, i)$; e potencial associação como uma associação potencial-definição-c-uso, uma potencial-associação-p-uso ou um potencial-du-caminho [MAL91a].

A extensão do grafo de fluxo de controle com informações de fluxo de dados (tipos de ocorrências de variáveis) consiste essencialmente em associar a cada nó i o conjunto $\text{defg}(i)$; o grafo estendido com informações de ocorrências de definição de variáveis é denominado **grafo def.**

A cada nó i tal que $\text{defg}(i) \neq \emptyset$ é associado um grafo denominado **grafo(i)** [MAL88b]; esse grafo contém todos os potenciais-du-caminhos com início no nó i . Cada nó k do grafo(i) é completamente identificado pelo número do nó no grafo de programa que lhe deu origem e pelo conjunto $\text{deff}(k)$ - o conjunto de variáveis definidas no nó i , mas que não são redefinidas num caminho do nó i para o nó k . Observe-se que $\text{deff}(k) \subseteq \text{defg}(i)$ e que $\text{deff}(i) = \text{defg}(i)$. Em [MAL88b] é apresentado um algoritmo para obter esses grafos; esses grafos são a base para a geração dos elementos requeridos e são utilizados na implementação da ferramenta **POKE-TOOL** [MAL89a, CHA91d] descrita no Capítulo 4.

Maldonado [MAL91a] introduziu a notação $[i, (j, k), \{v_1, \dots, v_n\}]$ para representar o conjunto de associações $[i, (j, k), v_1], \dots, [i, (j, k), v_n]$; ou seja, $[i, (j, k), \{v_1, \dots, v_n\}]$ indica que existe no grafo(i), pelo menos um caminho livre de definição c.r.a v_1, \dots, v_n do nó i para o arco (j, k) . Observe-se que podem existir, no grafo(i), outros caminhos livres de definição c.r.a algumas das variáveis v_1, \dots, v_n , mas que não sejam, simultaneamente, livres de definição para todas as variáveis v_1, \dots, v_n .

Diz-se que um caminho $\pi_1 = (i_1, \dots, i_k)$ está incluído em um conjunto Π de caminhos se e somente se Π contém um caminho $\pi_2 =$

(n_1, \dots, n_m) tal que $i_1 = n_j$, $i_2 = n_{j+1}$, ... $i_k = n_{j+k-1}$, para algum j , $1 \leq j \leq m-k+1$. Diz-se que π_1 está incluído em π_2 ou que π_1 é um sub-caminho de π_2 .

Um caminho completo π cobre uma associação potencial-definição-c-uso $[i, k, x]$, (respectivamente, uma associação potencial-definição-p-uso $[i, (j, k), x]$) se ele incluir um caminho livre de definição c.r.a x do nó i para o nó j (respectivamente do nó i para o arco (j, k)). O caminho π cobre um potencial-du-caminho π_1 se π_1 estiver incluído em π . Um conjunto Π de caminhos cobre uma potencial-associação se algum elemento do conjunto o fizer. Note-se que, se um conjunto de caminhos Π cobrir uma potencial-associação do nó i para o nó j (respectivamente do nó i para o arco (j, k)), então existe um caminho livre de definição c.r.a variável $x \in \text{defg}(i)$ do nó i para o nó j (respectivamente, para o arco (j, k)), que é incluído em Π .

Seja $\pi = (n_1, n_2, \dots, n_k)$ um potencial-du-caminho c.r.a x ; define-se a extensão a ciclo de (π, x) [MAL91a], como o conjunto de caminhos livres de definição c.r.a x do tipo $(\lambda_1, \lambda_2, \dots, \lambda_k)$ onde cada λ_i é um caminho de comprimento maior ou igual a 1 (um), com início e término no nó n_i . Deve-se observar que, para qualquer potencial-du-caminho π c.r.a x , π pertence à extensão a ciclo de (π, x) .

Define-se a extensão a ciclo para uma potencial-p-uso-associação $[i, (j, k), x]$ como sendo o conjunto de caminhos π_1 tal que π_1 é a uma extensão a ciclo de (π, x) e $\pi = (n_1, \dots, n_j, n_k)$ é um potencial-du-caminho de i para o arco (j, k) c.r.a x . Denota-se ainda, por L_p , para cada n_p , $1 < p \leq j$, o conjunto de caminhos livres de definição c. r. a x que começam e terminam em n_p . Para um particular potencial-du-caminho π , a extensão a ciclo de (π, x) é denotada por $(\pi, [i, (j, k), x])$, e representa o conjunto de caminhos é a extensão a ciclo de $[i, (j, k), x]$ gerados a partir de π . De uma forma resumida, a extensão a ciclo para $[i, (j, k), x] = (\pi/\pi_1) \in$ extensão a ciclo de (π, x) , onde π é um potencial-du-caminho c.r.a x do nó i para o arco (j, k) .

Além disso, denota-se por $\text{Pred}(np)$, o predicado associado a um nó np ; por $\text{Var_Pred}(np)$, as variáveis envolvidas no predicado do nó np ; por $f_{\text{aval}}(np, \pi)$ o resultado da avaliação do predicado do nó np no caminho π . Se $f_{\text{aval}}(np, \pi) = \text{true}$, significa que o predicado do nó np foi satisfeito para o caminho π ; caso contrário $f_{\text{aval}}(np, \pi) = \text{false}$.

Seja $\pi = (n_1, \dots, n_m)$ um caminho. Se n_p para $1 \leq p < m$ é um nó que possui um predicado, que deve ser avaliado simbolicamente, é necessário obter os valores para as variáveis do predicado, através da execução do caminho de i para p . A partir de um conjunto, associado ao nó p , de valores simbólicos para as variáveis do programa, pode-se definir a execução simbólica de um caminho em termos de apenas um nó. A partir do conjunto inicial, executam-se simbolicamente os comandos do primeiro nó e produz-se um novo conjunto de valores - computação simbólica das variáveis para o nó i no caminho π - que será utilizado como conjunto inicial para o nó $i+1$, de forma que, ao terminar a execução dos comandos do nó p , tem-se um conjunto que representa como o programa transformou os valores de entrada através do caminho π , computação simbólica das variáveis para o nó p no caminhos π , que será denotado $\text{CS}(p, \pi)$.

Um caminho completo é executável ou factível se existe um conjunto de valores que possa ser atribuído às variáveis de entrada, variáveis globais e parâmetros do programa, que causa a execução desse caminho; caso contrário, diz-se que ele é não executável [FRA87]. Um caminho é executável se ele for um subcaminho de um caminho completo executável, isto é, se ele for incluído em um caminho completo executável. Uma potencial-associação é executável se existir um caminho completo executável que cubra essa associação; caso contrário é não executável. Em função desses conceitos outros conjuntos são definidos: $f_{\text{pdcu}}(x, i) = \{ j \in \text{pdcu}(x, i) : \text{a potencial-associação-c-uso } (i, j, x) \text{ é executável}\}; f_{\text{pdpu}}(x, i) = \{(j, k) \in \text{pdpu}(x, i) : \text{a potencial-associação-p-uso } (i, (j, k), x) \text{ é executável}\}$. A partir desses conjuntos é que são caracterizados os critérios executáveis.

2.2 Caracterização e Previsão de Caminhos Não Executáveis.

Woodward, Hedley e Hennel [WOO 80, HED 85] realizaram um estudo com programas FORTRAN e os caminhos analisados foram formados pela concatenação de LCSAJ's ("Linear Code Sequence And Jump"). Um LCSAJ consiste de uma parte de código executado sequencialmente, que começa ou no início do programa ou em um ponto de desvio de fluxo de controle e termina em um outro ponto de desvio de fluxo de controle ou no final do programa.

Com base nesses componentes de programas foram propostos vários critérios de teste estrutural que diferem essencialmente na forma em que os LCSAJ's são concatenados, estabelecendo-se dessa maneira uma família de critérios de teste estrutural, usados como critérios de cobertura e que podem ser vistos como uma família de métricas que qualificam a atividade de teste do programa. Para avaliar a utilização dessas métricas foi realizado um estudo com um subconjunto de rotinas FORTRAN da biblioteca NAG - Numerical Algorithms Group. Em geral, os critérios propostos não puderam ser satisfeitos, pois a exemplo dos demais critérios de teste estrutural, ignoram a semântica dos programas e exigem combinações de LCSAJ's não executáveis.

Para implementar um sistema que auxiliasse a gerar dados de teste, Woodward [WOO 80] propôs o uso de alegações. Essas alegações são condições lógicas que devem ser satisfeitas por todos os caminhos gerados. Generalizações foram feitas considerando essas alegações; Hedley e Hennel [HED 85] apresentam uma análise sobre os caminhos não executáveis encontrados, suas principais causas e efeitos. Para isso foram selecionadas 88 rotinas da biblioteca NAG, que continham 4.790 LCSAJ's, numa média de 56 por rotina. Dentre esses, 12.5 % do total (619) foram considerados não executáveis. As causas da não executabilidade dos LCSAJ's puderam ser classificadas nas categorias abaixo; muitas podem ser identificadas estaticamente.

- a) 326 (52,7%): problemas com o número de vezes que o laço deve ser executado.

DO 20 J = 1,10

...

20 CONTINUE

ou *N = 2*

DO 40 J = 1,N

...

40 CONTINUE

- b) 79 (12.8%): laços que contêm condições com as variáveis dos laços.

DO 20 J = 1,N

IF (J .EQ. 1) ...

20 CONTINUE

- c) 143 (23,1%): teste de um valor imediatamente após sua definição.

ERROR = 0

IF (...) ERROR = 1

IF (...) ERROR = 2

IF (...) ERROR = 3

IF (ERROR .GT. 0) GO TO 300

- d) 57 (9.2%): teste de condições consecutivas, pois a construção *if then else* não está definida na versão do FORTRAN utilizada.

IF (MAX .GT. Q) ...

IF (MAX .LE. Q) ...

- e) 14 (2.3%) não puderam ser classificados em nenhuma dessas categorias.

Como uma possível solução para o problema de caminhos não executáveis Woodward, Hedley e Hennel sugerem o uso de técnicas de Programação Funcional para desenvolver programas que não possuam

caminhos não executáveis. Técnicas de programação funcional sugerem partitionar o domínio de entrada de um programa num número de subconjuntos associados com estruturas lógicas do programa e com requisitos funcionais, produzindo programas mais simples, com um número menor de caminhos, mais fáceis de testar, pois os dados de teste podem ser mais facilmente gerados.

Malevris, Yates e Veevers [MALE 90, YAT89], estudando a natureza dos caminhos não executáveis de um programa propuseram que o número de predicados envolvidos num programa possa ser considerado como uma métrica para prever não executabilidade, ou seja, quanto maior o número de predicados de um caminho, maior a probabilidade dele ser não executável.

Isso é mais ou menos intuitivo, considerando-se que se um caminho π possui $q > 0$ predicados, para que π seja executável é necessário que os predicados sejam consistentes e consistentes entre si. Se π_1 possui q predicados e π_2 possui r predicados, tal que $r > q$, π_2 tem maior probabilidade de ser não executável que π_1 .

Para verificar a validade dessas afirmações, Malevris realizou um estudo com 36 rotinas da biblioteca NAG e analisou 583 caminhos, obtendo a tabela a seguir. Esta tabela apresenta o número total de caminhos executáveis com o número de predicados q entre 1 e 16. Por exemplo, pode-se verificar que entre 185 caminhos executáveis, 21 caminhos apresentam número de predicados q igual a 4.

Malevris estudou a hipótese H_0 - de não existir forma de relacionamento entre o número de predicados do caminho e sua executabilidade -. Aos dados da tabela 2.1 foi aplicado um teste χ^2 . O teste mede o desvio da amostra para que a hipótese estudada seja verdadeira; $\chi^2_v(\alpha)$ é um valor padrão para cada integral $v > 0$, onde v é o número de graus de liberdade da amostra. Se $\chi^2 \geq \chi^2_v(\alpha)$, a hipótese pode ser rejeitada com probabilidade α . Malevris obteve $\chi^2 = 166.81$ e $\chi^2_{11}(0.005) = 26.775$ podendo dessa forma rejeitar a hipótese H_0 .

Tabela 2.1: Amostra utilizada por Malevris

no_pred	total	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
exec.	185	17	21	22	21	33	25	17	12	7	6	3	0	1	0	0	0
n exec.	398	4	9	13	24	20	45	52	38	69	47	28	13	10	5	17	4
no cam.	583	21	30	35	45	53	70	69	50	76	53	31	13	11	5	17	4

Investigando modelos estatísticos, obteve a equação $f_q = 1.0703e^{-0.1988q}$ com um resíduo (soma dos quadrados residuais, que demonstra se a equação obtida representa bem os pontos da amostra) suficientemente pequeno. Malevris considerou melhor o modelo exponencial, pois o modelo polinomial cresce ou diminui, não possuindo um limitante superior.

Malevris considerando que a executabilidade de um caminho é indecidível, encara a métrica número de predicados de um caminho como uma heurística. Baseado nessa heurística, Yates [YAT89], propõe uma estratégia para não gerar caminhos não executáveis, com o objetivo principal de reduzir esforço e tempo. A idéia é selecionar um conjunto Π , para realizar teste de ramos, que possui caminhos envolvendo um número mínimo de predicados, pois segundo a heurística, este seria o conjunto que tem maior probabilidade de conter um número menor de caminhos não executáveis.

2.3 Determinação de executabilidade

Rapps e Weyuker [RAP82, WEY84, RAP85] definiram uma família de critérios baseados em fluxo de dados. Os três critérios centrais dessa família são: **todas-defs**, **todos-usos**, **todos_du-caminhos**. Os outros três critérios **todos-p-usos**, **todos-p-usos/alguns c-usos**, **todos-c-usos/alguns p-usos** são variações do critério **todos-usos**. Esses critérios não satisfazem a propriedade da aplicabilidade; por isso, para cada critério C foi definido um critério C*, requerendo-se apenas caminhos e associações executáveis. Os critérios C* são referenciados como critérios executáveis [FRA86].

Para determinar se um conjunto de testes T satisfaz os critérios executáveis, é necessário examinar todas as associações exigidas pelo critério e verificar se são executáveis. Determinar se uma associação é ou não executável é indecidível pois é necessário saber quais caminhos são executáveis. Uma heurística baseada em análise de fluxo de dados foi proposta por Frankl [FRA87] e tem como objetivo determinar se uma associação é não executável. A associação será não executável se não existir caminhos executáveis que a cubram. A idéia é eliminar do conjunto de caminhos candidatos a cobrir a associação aqueles que não são executáveis e que possuam definições das variáveis da associação. A associação será possivelmente executável se o conjunto de candidatos viáveis não for vazio.

Se um nó de um caminho é um nó cabeça de laço cuja condição é avaliada verdadeira, é necessário, para sair do laço, que no caminho percorrido através deste laço tenha sido redefinida pelo menos uma das variáveis da condição para que essa torne-se falsa; além disso, o caminho precisa ser livre de definição com respeito às variáveis da associação. Para ilustrar a aplicação da heurística, considere o exemplo, extraído de [FRA 87], da Figura 2.1.

Considere a associação $(1, (2, 7), \text{done})$. Qualquer caminho que a cobre sai do nó 1, passa por um ou nenhum caminho através do laço do nó 2 e vai para o nó 7.

Como no nó 2 $\text{done} = \text{false}$, é necessário trocar o valor de done dentro do laço do nó 2; mas, como done é a única variável da condição e estamos procurando por caminhos livres de definição com relação a done , temos que a associação é não executável.

Para representar o conjunto de candidatos a cobrir uma associação, a heurística utiliza expressões de caminhos, e usa uma combinação de avaliação simbólica e informação de fluxo de dados para eliminar subconjuntos não executáveis.

A técnica é uma heurística no sentido de que ou responde "a associação é não executável" ou "a associação pode ser executável". Neste caso, a informação de que ela pode ser executável auxilia o usuário que pode determinar se ela é ou não executável e, se for o caso, selecionar um caso de teste que cubra a associação.

Frankl faz algumas restrições para os programas P avaliados. P não tem comandos repeat-until, as funções chamadas por P não têm parâmetros passados por referência, e os parâmetros formais são diferentes das variáveis globais, em P não ocorrem ponteiros.

Para representar os candidatos a cobrir as associações, Frankl define expressões regulares que representam conjuntos de caminhos. Para avaliar os predicados é definida uma técnica chamada Avaliação Simbólica Parcial para essas expressões de caminhos. Um algoritmo para implementar a heurística é também proposto por Frankl. Este algoritmo possui inúmeras restrições para ser implementado; entre elas, a exigência de um avaliador simbólico parcial de expressões e de um provador de teoremas.

```
1 begin
1   done := false;
2   while not done do
3     begin
3       S1;
3       if B then
4         done := true;
5       else
5         S2;
6       S3;
6     end;
7   end;
```

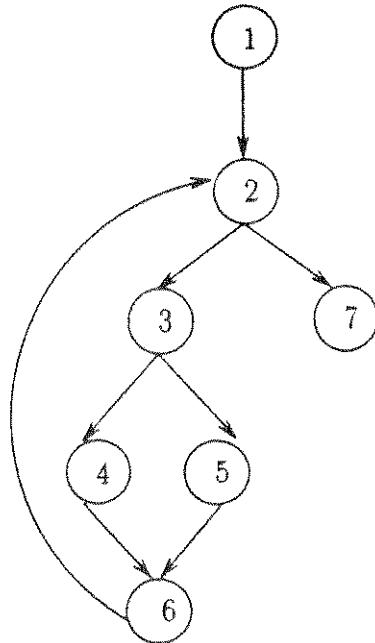


Figura 2.1 Exemplo da Heurística Proposta por Frankl

2.4 Critérios Potenciais Usos

Os critérios Potenciais Usos foram introduzidos por Maldonado, Chaim, Jino [MAL88a, MAL89b, MAL91a], e estão baseados no conceito de potencial uso, que diz: se uma definição de variável ocorre, existe um potencial uso a ela relacionado. Os critérios Potenciais Usos requerem basicamente que caminhos livres de definição, em relação a qualquer nó i que possua definição de variável e a qualquer variável x definida em i , sejam executados, independentemente de ocorrer uso dessa variável nesses caminhos. Neste sentido, pode-se verificar, por exemplo, que o valor de x não foi alterado nesses caminhos (possivelmente devido a efeitos colaterais) ganhando-se, desta forma, maior confiança de que a computação correta foi realizada; isto vem de encontro à filosofia discutida por Myers [MYE79]: um erro está claramente presente se um programa não faz o que supõe-se que ele faça, mas erros estão também presentes se um programa faz o que supõe-se que não faça; além disso, podem auxiliar a detectar dependências de fluxo de dados ausentes decorrentes, por exemplo, de erros de utilização de variáveis do programa.

Seja Π um conjunto de caminhos completos:

Critério Todos-potenciais-usos - Π satisfaz o critério todos-potenciais-usos se para todo nó i e para toda variável x para a qual existe uma definição em i , Π inclui pelo menos um caminho livre de definição c.r.a x do nó i para todo nó e para todo arco possível de ser alcançado a partir do nó i . Equivalentemente, em termos do conceito de potencial associação, Π deve cobrir todas as potenciais associações (i,j,x) : $j \in pdcu(x,i)$ e todas as potenciais associações $(i,(j,k),x)$: $(j,k) \in pdpu(x,i)$ para cada nó $i \in N$, $defg(i) \neq \emptyset$ e para cada $x \in def(i)$.

Critério Todos-potenciais-usos/du - Π satisfaz o critério todos-potenciais-uso/du se, para todo nó i e para toda variável x para a qual existe uma definição em i , Π inclui pelo menos um

potencial-du-caminho c.r.a. x do nó i para todo arco possível de ser alcançado a partir de i. Equivalentemente, em termos do conceito de potencial associação, II deve incluir, para cada $i \in N / \text{def}(i) \neq \emptyset$, um potencial-du-caminho de i para j c.r.a x para todas as potenciais associações $(i,j,x) / j \in \text{pdcu}(x,i)$ e um potencial-du-caminho de i para (j,k) c.r.a x para todas as potenciais associações $(i,(j,k),x) / (j,k) \in \text{pdpu}(x,i)$.

Critério Todos-potenciais-du-caminhos - II satisfaz o critério todos-potenciais-du-caminhos se para todo nó i e para toda variável x para a qual existe uma definição em i, II inclui todos os potenciais-du-caminhos c.r.a x em relação ao nó i para todo nó $j \in \text{pdcu}(x,i)$ e para todo arco $(j,k) \in \text{pdpu}(x,i)$. Equivalentemente, em termos do conceito de potencial associação, II deve incluir para cada $i \in N : \text{defg}(i) \neq \emptyset$, todos os potenciais-du-caminhos de i para j c.r.a cada variável $x \in \text{defg}(i)$ para todas as potenciais associações $[i,j,x] : j \in \text{pdcu}(x,i)$ e todos potenciais-du-caminhos de i para (j,k) c.r.a cada variável $x \in \text{defg}(i)$ para todas as potenciais associações $[i,(j,k),x] : (j,k) \in \text{pdpu}(x,i)$.

Também os critérios Potenciais-Usos foram modificados para satisfazer a propriedade da aplicabilidade e definidos novos critérios denominados Critérios Potenciais Usos Executáveis - todos os potenciais usos executáveis, todos os potenciais usos/du executáveis, todos os potenciais du-caminhos executáveis. Então, para cada critério Potencial Uso C define-se um novo critério C', onde a modificação consiste em selecionar as potenciais associações requeridas utilizando-se os conjuntos $\text{fpdcu}(x,i)$ e $\text{fpdpu}(x,i)$; apenas elementos executáveis, portanto.

Critério (todos-potenciais-usos) - II satisfaz o critério (todos-potenciais-usos) se, para todo nó $i \in N / \text{defg}(i) \neq \emptyset$ e para toda variável x / $x \in \text{defg}(i)$, II incluir todas as potenciais associações $[i,j,x] / j \in \text{fpdcu}(x,i)$ e todas as potenciais associações $[i,(j,k),x] / (j,k) \in \text{fpdpu}(x,i)$.

Criterio (Todos-potenciais-usos/du) - Π satisfaz o critério (todos-potenciais-usos/du) se, para todo nó $i \in N / defg(i) \neq \emptyset$ e para toda variável $x / x \in defg(i)$, Π incluir um potencial-du-caminho executável para todas as potenciais associações $[i,(j,k),x] / j \in fpdcu(x,i)$ e um potencial du-caminho executável para todas as potenciais associações $[i,(j,k),x] / (j,k) \in fpdpu(x,i)$.

Critério (Todos-potenciais-du-caminhos) - Π satisfaz o critério (todos-potenciais-du-caminhos), se para todo nó $i \in G / defg(i) \neq \emptyset$, Π incluir todos os potenciais-du-caminhos executáveis c.r.a todas as variáveis $x \in defg(i)$ a partir do nó i para todo nó $j \in fpdpu(x,i)$ e para todo arco $(j,k) \in fpdpu(x,i)$. Equivalentemente, em termos do conceito de potencial associação, Π deve incluir todos os potenciais-du-caminhos executáveis de i para j c.r.a cada variável $x \in defg(i)$, para todas as potenciais associações executáveis $[i,j,x] / j \in fpdcu(x,i)$ e todos os potenciais-du-caminhos executáveis de i para (j,k) c.r.a cada variável $x \in defg(i)$ para todas as potenciais associações executáveis $[i,(j,k),x] / (j,k) \in fpdpu(x,i)$.

critério (todos-nos) - Requer a execução de todos os nós executáveis.

critério (todos-ramos) - Requer a execução de todos os ramos executáveis.

critério (todos-caminhos) - Requer a execução de todos os caminhos executáveis.

O critério todos-potenciais-du-caminhos foi estendido utilizando-se o conceito de extensão a ciclo com o objetivo principal de obter-se uma hierarquia de critérios que inclusse os critérios todos-ramos e todas-definições, mesmo na presença de caminhos não executáveis; observe-se que o critério todos-potenciais-usos é na realidade o critério todos-potenciais-usos/du estendido a ciclo. Os critérios Potenciais Usos Básicos e os critérios Potenciais-Usos

Estendidos a Ciclo constituem a Família de Critérios Potenciais Usos. Para essa família também foi derivada a correspondente Família de Critérios Potenciais Usos Executáveis [MAL 91a].

Critério Todos-Potenciais-usos/du Estendido a Ciclo- II satisfaz o critério todos-potenciais-usos/du estendido a ciclo para um dado programa P, se e somente se, II incluir, para cada potencial-associação $[i,j,x] / j \in pdcu(x,i)$ e para cada potencial-associação $[i,(j,k),x] / j \in pdpu(x,i)$, algum caminho $\pi_1 \in$ extensão-a-ciclo(π,x), onde π é um potencial-du-caminho c.r.a.x do nó i para o nó j ou para o arco (j,k) respectivamente.

Critério Todos-potenciais-du-caminhos Estendido a Ciclo - II satisfaz o critério todos-potenciais-du-caminhos estendido a ciclo se e somente se, para todo nó $i \in G / defg(i) \neq \emptyset$, II incluir para cada variável $x \in defg(i)$ e para cada potencial-du-caminho π c.r.a x a partir de i, algum caminho $\pi_1 \in$ extensão a ciclo(π,x).

Critério (Todos-potenciais-usos/du Estendido a Ciclo)- II satisfaz o critério todos-potenciais-usos/du estendido executável para um dado programa P, se e somente se, II incluir, para cada potencial-associação executável $[i,j,x] / j \in fpdcu(x,i)$ e para cada potencial-associação executável $[i,(j,k),x] / j \in fpdpu(x,i)$, algum caminho executável $\pi_1 \in$ extensão a ciclo(π,x), onde π é um potencial-du-caminho c.r.a x do nó i para o nó j ou para o arco (j,k), respectivamente.

Critério (Todos-potenciais-du-caminhos Estendido a Ciclo)- II satisfaz o critério todos potenciais-du-caminhos estendido a ciclo executável se e somente se para todo nó $i \in G / defg(i) \neq \emptyset$, II incluir para cada variável $x \in defg(i)$ e para cada potencial-du-caminho π c.r.a x a partir de i, algum caminho executável $\pi_1 \in$ extensão a ciclo(π,x).

Estudos comparativos entre os critérios baseados em fluxo de dados têm sido conduzidos apoiados principalmente por uma relação de inclusão e pelo estudo da complexidade dos critérios [CLA86, RAP85,

WEY84, NAT88, MAL89b, MAL91a, MAL91b]. A complexidade de um critério C é definida como o número máximo de casos de teste requerido pelo critério no pior caso; ou seja, dado um programa qualquer P, se existir um conjunto de casos de teste T que seja C-adequado para P, então existe um conjunto de casos de teste T₁, tal que a cardinalidade de T₁ é menor ou igual a complexidade do critério C. Maldonado [MAL91a] mostra que todos os critérios de teste baseados em fluxo de dados têm complexidade de ordem exponencial.

A relação de inclusão estabelece uma ordem parcial entre os diversos critérios; portanto, como resultado da análise de inclusão, obtém-se uma ordem parcial entre esses critérios. Diz-se que um critério de teste C₁ inclui um critério de teste C₂ se, para qualquer grafo de fluxo de controle G (qualquer programa P), qualquer conjunto de caminhos completos T que seja C₁-adequado para P seja C₂-adequado para P; ou seja, se T satisfaz C₁ também satisfaz C₂. O critério C₁ inclui estritamente um critério C₂, denotado por C₁ => C₂, se C₁ inclui C₂ e para algum grafo G existe um conjunto de caminhos completos T de G que satisfaz C₂ mas não satisfaz C₁. Se nem C₁ => C₂ nem C₂ => C₁, diz-se que os critérios são incomparáveis.

A presença de caminhos não executáveis modifica consideravelmente algumas das propriedades dos critérios; por exemplo, Frankl [FRA87, FRA88] observou que a relação de inclusão entre os critérios da Família de Critérios Fluxo de Dados muda significativamente: nenhum deles inclui o critérios todos os ramos [WEY86, WEY88a], por exemplo, não preenchendo algumas das propriedades esperadas de um bom critério de teste [CLA86, MAL91a]. Diz-se que esses critérios não estabelecem uma "ponte" ("bridge the gap") entre os critérios todos-ramos e todos os caminhos.

A relação de inclusão dos critérios apresentados nesta seção e os critérios de Rapps e Weyuker, pode ser vista nas Figuras 2.2 e 2.3 [MAL91a]. Os critérios Potenciais Usos incluem os critérios baseados em análise de fluxo de dados, além de possuirem a mesma complexidade; e os

critérios Potenciais Usos incluem o critério todos os ramos mesmo na presença de caminhos não executáveis. Além de outros aspectos, os critérios Potenciais Usos são os únicos critérios de fluxo de dados que têm essa característica e nenhum outro critério de fluxo de dados inclui os critérios Potenciais Usos.

Analogamente aos outros critérios, os critérios Potenciais Usos necessitam de uma ferramenta para sua aplicação efetiva. Para tanto, foi implementada uma ferramenta de teste, denominada POKE-TOOL, com o objetivo de viabilizar a realização de comparações entre esses critérios e os demais critérios de teste estrutural, bem como a avaliação da adequação desses critérios a certas classes de erros. O Capítulo 3 apresenta uma descrição resumida da POKE-TOOL, maiores detalhes poderão ser obtidos em [MAL89a,CHA91d].

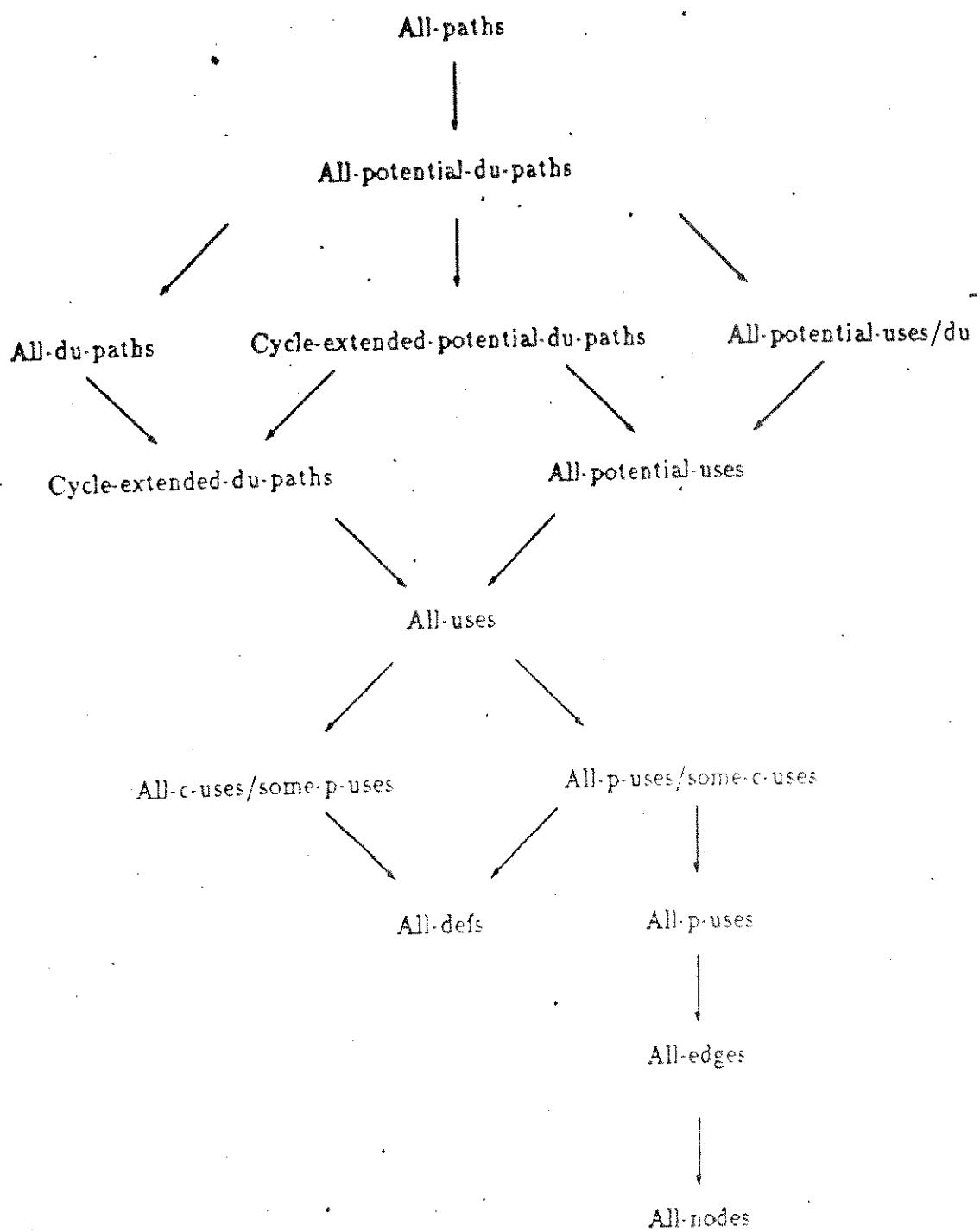


Figura 2.2: Ordem Parcial entre a Família de Critérios Potenciais Usos e a Família de Critérios de Fluxo de Dados

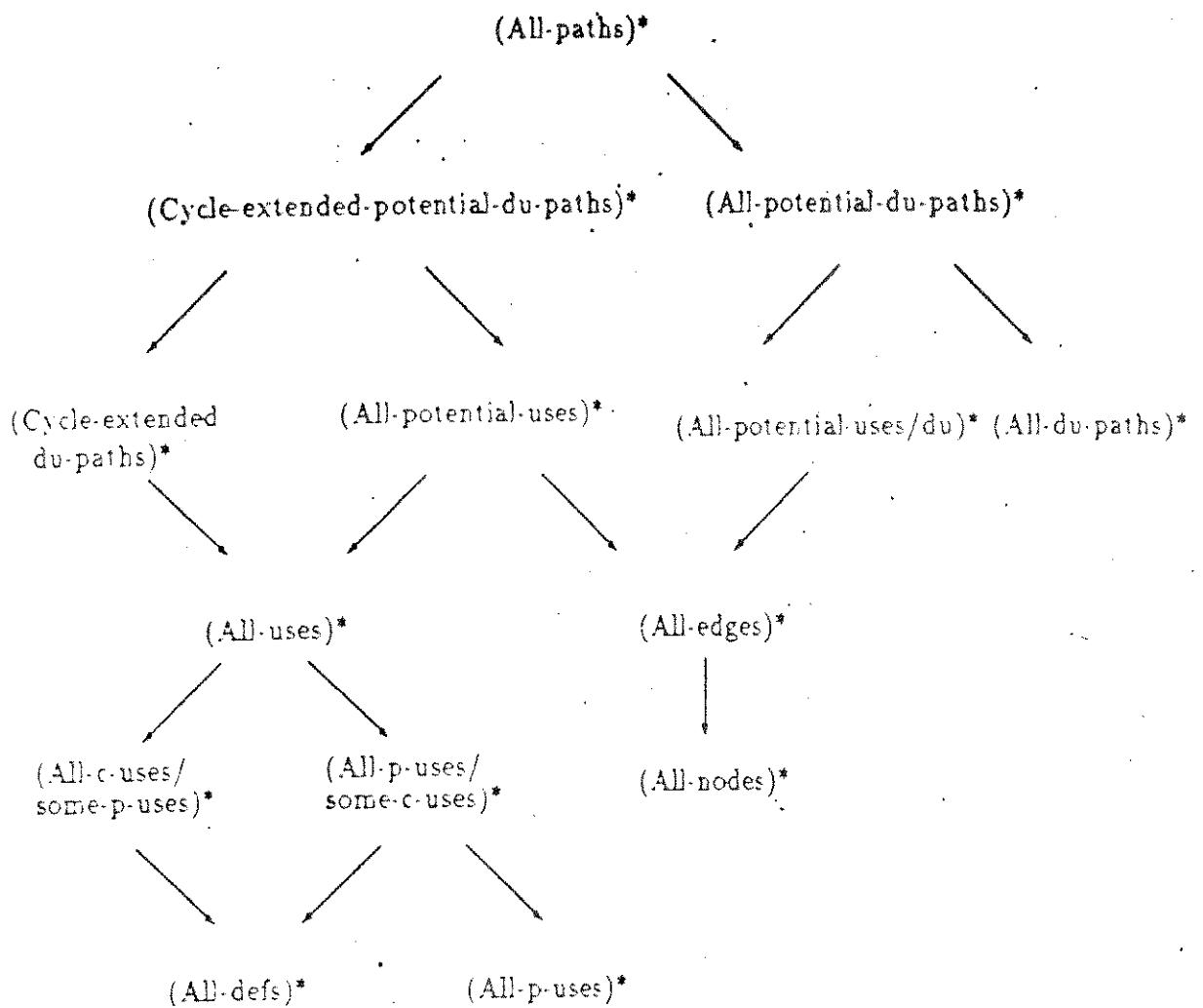


Figura 2.3: Ordem Parcial entre a Família de Critérios Potenciais Usos e a Família de Critérios de Fluxo de Dados na Presença de Caminhos não Executáveis

2.5 Considerações Finais

Esse capítulo apresentou a terminologia utilizada nesse trabalho e uma síntese dos trabalhos relativos a não executabilidade de caminhos nas atividades de teste. Os trabalhos descritos objetivam caracterizar, prever e determinar não executabilidade.

Woodward, Hedley e Hennel apresentam quais as principais causas de não executabilidade encontradas num subconjunto de rotinas FORTRAN. Caracterizar caminhos não executáveis é muito importante para saber como prever e aplicar heurísticas que tenham a maior probabilidade de determinar quais caminhos são não executáveis.

Por isso, durante a condução do "benchmark", realizado por Hedley e Hennel, as causas de não executabilidade foram anotadas e são apresentadas no capítulo seguinte. Também no capítulo seguinte são apresentados outros resultados do "benchmark": um estudo sobre a influência de várias características do programa no número de caminhos não executáveis; alguns modelos de estimativa, a exemplo da experiência de Malevris, apresentada na Seção 2.2 deste capítulo, foram obtidos estudando-se a influência do número de predicados de potenciais-du-caminhos na sua executabilidade.

Para determinar não executabilidade foi apresentada uma heurística proposta por Frankl, que está voltada para laços nos caminhos e utiliza informação sobre o fluxo de dados e técnicas de avaliação simbólica.

O capítulo apresentou também apresentou uma síntese dos critérios Potenciais Usos introduzidos por Maldonado, Chaim e Jino cuja aplicação é apoiada pela ferramenta POKE-TOOL. A essa ferramenta foram incorporadas facilidades para tratamento de caminhos não executáveis, entre elas a heurística proposta por Frankl. Essas facilidades e seus aspectos de implementação são discutidos no Capítulo 4.

CAPÍTULO 3

RESULTADOS DA APLICAÇÃO DE UM "BENCHMARK"

Como foi dito no Capítulo 1, estudos têm sido conduzidos com o objetivo de investigar o custo da aplicação de critérios de teste baseados em análise de fluxo de dados. Um conjunto de 29 rotinas, extraído do livro de Kernighan & Plauger [KER81], "Software Tools in Pascal", foi utilizado por Weyuker [WEY90] para avaliar o custo da aplicação de sua família de critérios. Hedley [HED85] e Malevris [MALE90] utilizaram um conjunto de rotinas FORTRAN para estudar o problema de não executabilidade caracterizando as principais causas que geram caminhos não executáveis e métricas para estabelecer modelos de previsão de executabilidade de caminhos através de programas.

Na mesma linha dos trabalhos citados, foi conduzido o mesmo "benchmark" proposto por Weyuker, utilizando-se a ferramenta POKE-TOOL, com o objetivo de determinar o custo e a factibilidade dos critérios Potenciais Usos, e estudar a influência de várias características do programa em modelos de estimativa do número de casos de teste requeridos para satisfazer esses critérios [MAL91a,MAL91c].

Neste capítulo apresentam-se os principais resultados obtidos do ponto de vista de executabilidade de caminhos durante a condução do "benchmark". Primeiramente, será dada uma descrição de como foi realizado o experimento e a coleta de dados. Em seguida, serão apresentados os resultados do estudo sobre as principais causas de não executabilidade nas rotinas testadas. São também apresentados os resultados e modelos estatísticos obtidos analisando a influência de várias características do programa sobre o número de caminhos não executáveis e, adicionalmente, como feito por Malevris, analisando o

relacionamento entre o número de predicados de potenciais-du-caminhos e a executabilidade desses elementos.

A partir das observações desse experimento são também propostas as facilidades a serem incorporadas à POKE-TOOL para tratamento de não executabilidade, discutidas no Capítulo 4.

3.1 Realização e Coleta de Resultados

Descreve-se a seguir a estratégia adotada para a condução do "benchmark" [MAL91a]. Inicialmente, para cada um dos programas que compõem o "benchmark", foram determinados os seguintes dados (características do programa), sintetizados na tabela 3.1:

- número de comandos de decisão (C1).
- número de variáveis utilizadas (C2).
- número de definições de variáveis (C3).
- número de nós com definição de variáveis (C4).
- número de nós do grafo de programa (C5).

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 disponíveis na POKE-TOOL - todos potenciais-usos, todos potenciais-usos/du e todos potenciais du-caminhos, referenciados como Critérios Potenciais Usos Básicos. Abaixo estão listadas as principais etapas que foram seguidas:

- leitura da descrição funcional da unidade;
- implementação da unidade na linguagem C;
- elaboração do conjunto de casos de teste, baseada apenas nos aspectos funcionais da unidade sem, entretanto, aplicar nenhum critério de teste funcional;
- submissão dos casos de teste iniciais à POKE-TOOL;
- análise do conjunto de casos de teste inicial em relação aos

critérios:

- elaboração de novos casos de teste e submissão à POKE-TOOL com o objetivo de cobrir os elementos restantes. Se o elemento foi constatado como não executável, a causa de sua não executabilidade foi anotada. O procedimento continuou até que todos os elementos executáveis fossem cobertos.

Com o propósito de avaliar a hipótese de Malevris para o benchmark testado, também foram realizados para cada rotina o cálculo do número de predicados dos caminhos executados (executáveis) e dos que não puderam ser executados (não executáveis).

Como resultado imediato da realização do "benchmark" foram obtidos os seguintes resultados sintetizados na tabela 3.2, Onde aparecem:

- Conjunto Inicial dos Casos de Teste (CICT) e análise da adequação do CICT para cada um dos critérios.
- Conjunto de Casos de Teste para o Critério CR1 (CFCT/PU).
- Conjunto de Casos de Teste para o Critério CR2 (CFCT/PDU).
- Conjunto de Casos de Teste para o Critério CR3 (CFCT/PDU).
- Conjunto de associações e caminhos não executáveis para os critérios Potenciais Usos.

O número total (porcentagem) de elementos não executáveis para cada critério foi maior que o esperado. Além disso, note-se que apenas 3 dos programas do benchmark --- DODASH, ARCHIVE e GETCMD --- não apresentam elementos não executáveis. Deve-se ressaltar também que dois dos programas testados --- AMATCH e RQUICK --- são recursivos. Como a versão atual da POKE-TOOL não trata uniformemente procedimentos recursivos e procedimentos não recursivos, as análises foram conduzidas considerando-se dois conjuntos distintos de programas, um envolvendo as unidades recursivas, denotado por TOTAL, e outro excluindo-as, denotado por TOTAL-R.

Tabela 3.1: Variáveis de Controle Relativas ao "Benchmark"

UNIDADE	# Comandos de Decisão / # nós (C1/C5)	# Variáveis Utilizadas (C2)	# Definições de variáveis (C3)	# Nós com Definição de Variáveis (C4)
DODASH	6/19	7	8	3
ARCHIVE	6/18	4	9	6
RQUICK	6/14	11	14	6
GETFNS	6/17	6	13	9
COMPARE	6/14	9	16	5
ENTAB	6/15	4	10	8
EXPAND	6/18	2	6	5
CMP	5/16	3	5	2
COMPRESS	5/16	3	6	5
UNROTATE	7/20	5	21	13
TRANSLIT	12/30	8	22	12
COMMAND	15/19	29	44	15
GETCMD	14/43	2	16	15
AMATCH	7/22	6	19	12
OMATCH	15/29	5	12	8
GTEXT	3/9	9	13	5
GETDEF	9/26	7	12	7
GETONE	9/22	7	8	3
GETNUM	6/19	4	4	1
MAKEPAT	10/30	8	22	13
SPREAD	5/14	9	18	8
CHANGE	5/11	4	12	7
GETFN	5/13	5	7	4
SUBST	9/27	17	36	16
EDIT	9/22	7	16	10
GETLIST	6/16	9	19	8
APPEND	5/16	6	12	8
CKGLOB	6/19	6	11	8
OPTPAT	5/15	3	5	3

**Tabela 3.2: Variáveis Respostas; Informação Dinâmica
Relativa às Rotinas do "Benchmark"**

UNIDADE	Card. CICT	Cardinalidade CFCT			# Assoc. não exec./requeridas		
		CR1	CR2	CR3	CR1	CR2	CR3
DODASH	7	15	15	15	0/33	0/33	0/21
ARCHIVE	7	7	7	7	0/17	0/17	0/17
RQUICK	1	2	2	2	6/55	19/55	33/51
GETFNS	5	7	7	7	8/44	14/44	9/34
COMPARE	10	11	12	12	0/38	6/38	30/64
ENTAB	8	14	14	14	23/80	31/80	41/70
EXPAND	12	12	13	13	14/39	15/39	10/25
CMP	7	7	7	7	3/13	3/13	3/12
COMPRESS	12	14	14	14	3/39	6/39	10/34
UNROTATE	3	6	6	6	14/94	39/94	29/70
TRANSLIT	33	37	38	48	6/138	6/138	203/333
COMMAND	15	15	15	15	3/47	3/47	19/61
GETCMD	15	15	15	15	0/119	0/119	0/119
AMATCH	9	14	14	14	70/112	70/112	61/87
OMATCH	13	13	13	13	12/33	12/33	25/43
GTEXT	5	5	5	5	6/33	11/33	10/22
GETDEF	13	20	23	23	3/59	9/59	9/49
GETONE	8	12	14	20	3/62	6/62	147/177
GETNUM	7	8	8	8	0/8	0/8	5/12
MAKEPAT	36	39	39	39	73/207	87/207	169/253
SPREAD	7	12	17	17	7/61	10/61	14/47
CHANGE	8	8	8	8	1/38	1/38	2/27
GETFN	7	7	7	7	2/18	2/18	7/20
SUBST	10	29	26	28	43/219	74/219	235/322
EDIT	17	32	40	40	20/130	21/130	261/357
GETLIST	6	13	14	15	20/82	22/82	64/102
APPEND	7	7	7	7	11/49	12/49	22/43
CKGLOB	7	7	7	7	1/35	11/35	7/27
OPTPAT	5	5	5	5	1/13	1/13	13/21

Nas Tabelas 3.3 e 3.4 estão sintetizados os dados --- números de predicados nos potenciais-du-caminhos --- para as rotinas do benchmark, com e sem as unidades recursivas, respectivamente; no Apêndice A esses dados estão organizados em tabelas, uma para cada rotina.

Tabela 3.3: 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	143
exc	19	106	112	103	122	113	122	101	55	47	61	33	25	16	10	17	8	5	2	107
tot	19	125	153	147	212	231	317	348	186	162	199	150	82	47	34	57	22	10	8	250

Tabela 3.4: Número de Predicados Contidos nos Potenciais-du-caminhos das Rotinas do Benchmark Excluindo-se as Unidades Recursivas

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	to
nex	0	19	40	37	83	109	190	228	105	109	130	111	57	31	24	40	14	5	6	133
exc	19	104	104	101	115	106	117	96	50	44	61	33	25	16	10	17	8	5	2	103
tot	19	123	144	138	198	215	307	324	155	153	191	144	82	47	34	57	22	10	8	237

3.2 Caracterização dos caminhos não executáveis.

A exemplo do trabalho de Hedely 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 são sempre verdadeiras e que serão sempre executados, ou com laços que contêm teste de variáveis que controlam laços e que serão satisfeitos apenas uma vez durante a repetição do laço. Veja o exemplo dado abaixo (Figura 3.1). O caminho dado por (1,2,6) é não executável pois, na entrada do laço a condição do comando *for* é sempre verdadeira. O caminho (1,2,3,5) também é não executável, pois na primeira execução do laço a condição do nó 3 é verdadeira e o caminho executado será obrigatoriamente (1,2,3,4, ...).

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 3.2). Note 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 próximo nó executado o 19.

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, pois existiriam inconsistências nos predicados que caracterizam o caminho.

Na rotina `getfns` (Figura 3.3), os dois laços em sequência possuem o mesmo predicado; portanto, os caminhos $(3,4,6,7,8,7)$ e $(4,5,4,6,7,9)$ são não executáveis. Por outro lado os predicados dos laços aninhados são dependentes pois $i < nfiles - 1 = j$ (que vale $i+1 < nfiles$, e o caminho $(10,11,12,16)$ é não executável.

```

1      ....
1,2,5  for(i=0; i<=5; i++)
3      {
3          if (i==0)
4              printf("Msgem");
5      }
6      ....

```

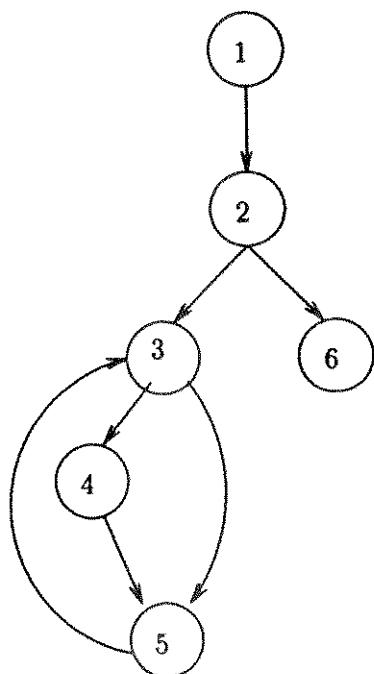


Figura 3.1 Caminhos não executáveis gerados por laços que são sempre executados.

```

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;
19  else
19   *status = OK;
20 }....

```

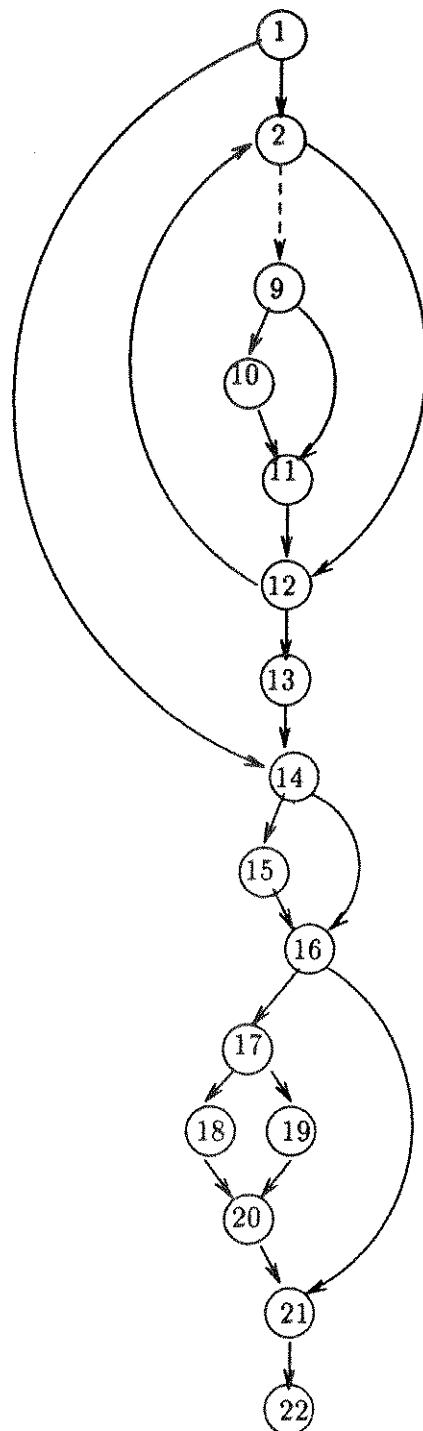


Figura 3.2 Rotina GETONE

```

void getfns()
1 { errcount = 0;
1   nfiles = nargs() - 2;
1   if (nfiles > MAXFILES)
2     error("archive too many files");
3,4,5 for (i = 0; i<nfiles; i++)
5   getarg(i+3,fname[i]);
6,7,8 for (i = 0;i<nfiles; i++)
8   fstat[i] = 0;
9,10,16 for (i = 0;i<nfiles-1; i++)
11,12,15for (j=i+1; j<nfiles; j++)
13   if (equal(fname[i],fname[j])
14   {
14     puts(fname[i]);
14     error("duplicate file name");
14   }
15 } ....

```

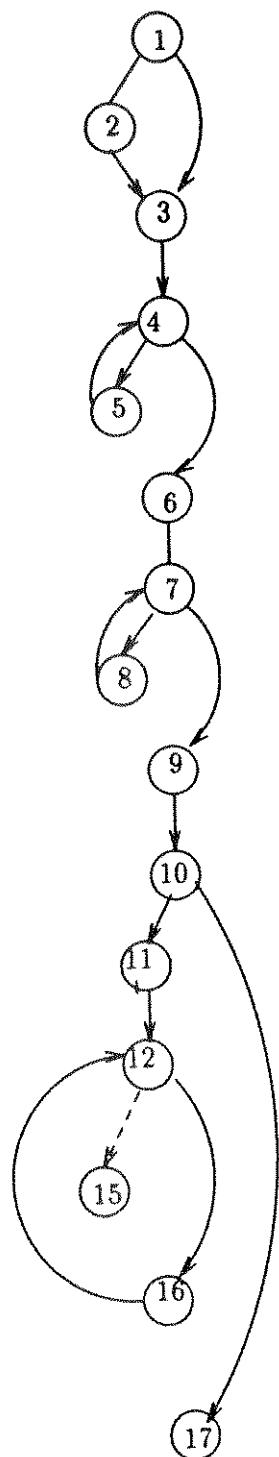


Figura 3.3 Rotina GETFNS

Na rotina `command` (Figura 3.4), 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)`.

```

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 }

```

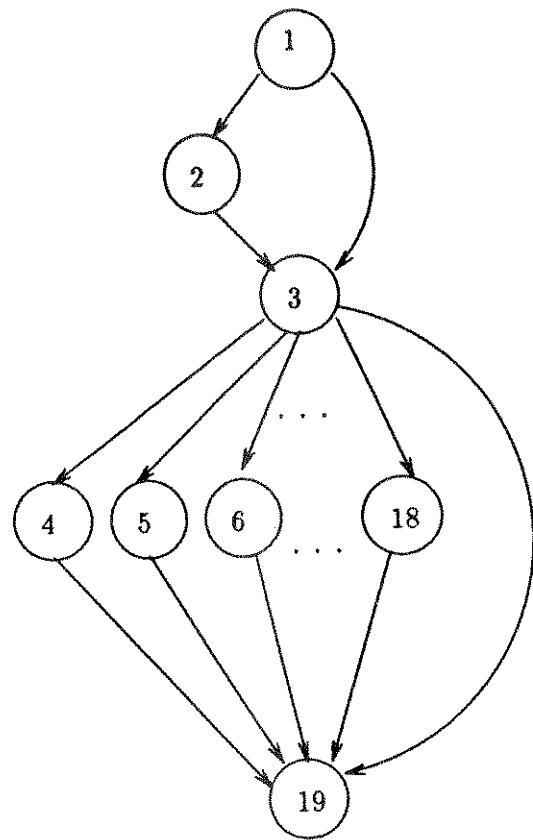


Figura 3.4 Rotina COMMAND

c. Teste de variáveis imediatamente após a sua definição: Como exemplo desta situação, novamente na rotina `getone` (Figura 3.2), no nó 15 `*status = ERR;` esse valor torna o predicado do nó 16 falso 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 acima. Muitas são combinações dos quatro casos apresentados, outras são casos isolados.

Deve-se ressaltar que não foi contabilizada a porcentagem de ocorrência dessas categorias nas rotinas testadas; porém, os casos a e c foram os que mais ocorreram nas rotinas testadas, muitos deles envolvendo laços são cobertos pela heurística de Frankl. Os caminhos mais difíceis de serem analisados foram os caminhos que possuíam 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 17 na rotina `translit` (Figura 3.5).

Para avaliar tais predicados, foi importante determinar o que podia ser assumido como verdadeiro em cada nó. Por exemplo, se o caminho que está sendo analisado contém o nó 4, então `allbut = true`, no nó 16, e `squash` receberá o valor `true`. Isto é um fato que pode ser assumido como verdadeiro e utilizado para avaliar o predicado do nó 17.

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. Estas facilidades, que foram necessárias para determinação manual dos caminhos não executáveis, foram incorporadas à `POKE-TOOL`, conforme proposto no Capítulo 4.

```

void translit()
1 { if (!getarg(1,arg))
.. ....
3. if(allbut)
4     l = 1;
5 else
5     l = 0;
.....
16 ....
16 squash = ((strlen(fronset)>lastto+1))||(allbut));
17 do{
17 l = xIndex(fromset,c=getchar(),allbut,lastto);
17 if ((saquash) && (l>=lastto) && (lastto+l>0))
18(
18 putchar(toset[lastto]);
19 do
19 l = xIndex(fromset,c=getchar(),allbut,lastto);
20     while(l>=lastto);
21 }
22 ....

```

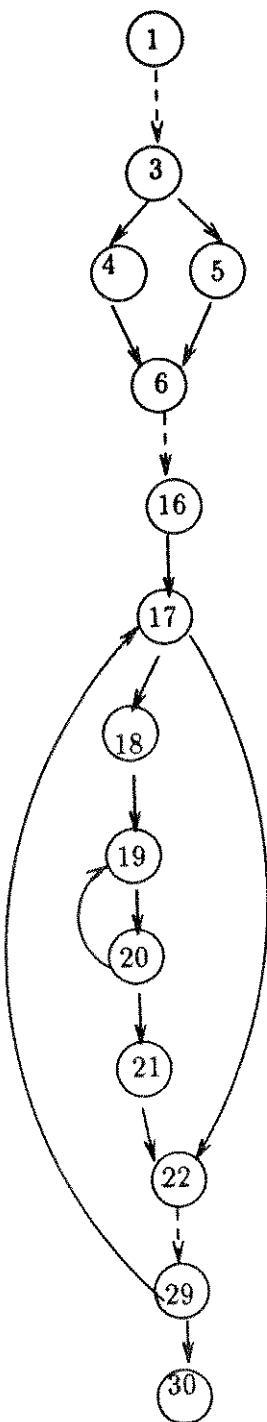


Figura 3.5 Rotina Translit

3.3 Características do Programa X Não Executabilidade.

Realizou-se uma análise exploratória da influência de várias características do programa no número de casos de teste exigidos pelos critérios Potenciais Usos Básicos e na determinação de caminhos não executáveis. Foram explorados vários modelos com o apoio do sistema MINITAB [RYA76] - um sistema de Cálculos Estatísticos - utilizando-se as variáveis descritas na Tabela 3.2. As variáveis resposta (ou dependentes) *nct* e *nex*, representam respectivamente o número de casos de teste exigido e o número de caminhos não executáveis [MAL91a, MAL91c].

Maldonado [MAL91a] mostra, através de análise estatística, que o número de casos de teste requeridos para satisfazer os critérios Potenciais-Usos é linear em relação ao número de comandos de decisão do programa em teste. Mostra ainda que outras características do programa, como número de variáveis utilizadas, levam a melhores modelos de estimativa. Alguns desses modelos são ilustrados na Tabela 3.5. Nessa tabela estão também ilustrados os modelos obtidos para estimar o número de caminhos não executáveis, utilizando-se as mesmas variáveis de controle propostas por Maldonado, para os três critérios Potenciais Usos. No Apêndice A são apresentados, de forma mais detalhada, os modelos para estimar o número de caminhos não executáveis, obtidos para o critério todos-potenciais-du-caminhos utilizando-se as variáveis de controle --- *variav* (número de variáveis), *def* (número de definições), *nodef* (número de nós com definições) --- e o número de potenciais-du-caminhos.

Deve-se ressaltar que todas as variáveis de controle têm também forte influência na determinação dos modelos de estimativa do número de caminhos não executáveis. Por exemplo, o número de nós com definições de variáveis, *nodef* é bastante significativo na determinação do modelo para estimativa do número de associações não executáveis para o critério CR1, até mesmo mais que o número de comandos de decisão.

Um outro ponto que merece ser ressaltado é a forte correlação do número de potenciais du-caminhos com o número de caminhos e associações não executáveis para o critério todos-potenciais-du-caminhos, tendo sido determinados modelos com r^2 da ordem de 99%, onde r^2 representa o quadrado do coeficiente de correlação amostral r , e indica a proporção de variabilidade explicada pelo modelo. Segundo Maldonado [MAL91a], o conceito de potencial uso e a consequente definição de potencial-du-caminho, fornecem uma medida do grau de inter-relacionamento entre a estrutura de controle do programa e o fluxo de dados que pode refletir outras propriedades associadas ao programa: dificuldade de testar; numero de caminhos não executáveis; dificuldade de manutenção e de depuração; entre outras.

Tabela 3.5: Melhores Modelos Obtidos Para os Critérios Potenciais Usos

Critério todos-potenciais-usos (CR1)

Número de casos de teste (nct)	Associações não executáveis (nex)
$nct = -11.7 + 3.96 t$	$nex = -25.6 + 5.82 t$
$nct = -13.4 + 3.85 t + 0.411$ variav	$nex = -15.5 + 2.46 t + 1.52$ variav
$nct = -12.6 + 3.64 t + 0.247$ def	$nex = -11.6 + 1.35 t + 0.979$ def
$nct = -13.2 + 3.31 t + 0.868$ nodef	$nex = -25.7 + 2.69 t + 3.03$ nodef
$nct = 6.90 + 0.0759$ potducam	$nex = 2.67 + 0.0888$ potducam

Critério todos-potenciais-usos/du (CR2)

Número de casos de teste (nct)	Associações não executáveis (nex)
$nct = -13.8 + 4.35 t$	$nex = -33.3 + 7.79 t$
$nct = -14.9 + 4.06 t + 0.454$ variav	$nex = -27.5 + 4.08 t + 2.72$ variav
$nct = -13.5 + 3.72 t + 0.288$ def	$nex = -20.3 + 1.93 t + 1.82$ def
$nct = -13.5 + 3.72 t + 0.534$ nodef	$nex = -19.1 + 1.05 t + 3.94$ nodef
$nct = 7.21 + 0.0757$ potducam	$nex = 2.77 + 0.160$ potducam

Critério todos potenciais-du -caminhos (CR3)

Número de casos de teste (nct)	Caminhos não executáveis (nex)
$nct = -16.3 + 4.86 t$	$nex = -152 + 31.6 t$
$nct = -17.4 + 4.58 t + 0.454$ variav	$nex = -173 + 26.7 t + 8.37$ variav
$nct = -16.4 + 4.23 t + 0.333$ def	$nex = -154 + 23.6 t + 4.33$ def
$nct = -16.5 + 4.24 t + 0.624$ nodef	$nex = -153 + 23.9 t + 7.66$ nodef
$nct = 7.04 + 0.0976$ potducam	$nex = -14.3 + 0.766$ potducam

3.4 Número de Predicados X Executabilidade de caminhos

Segundo Malevris [MALE 90], o número de predicados de um caminho influí na sua executabilidade. Estudos foram realizados com o objetivo de validar suas afirmações para as rotinas do "benchmark", verificando-se o tipo de relacionamento existente entre o número de predicados e a executabilidade de um potencial du-caminho dado.

Primeiramente foram coletados, para cada rotina, o número de caminhos executáveis e não executáveis, e o número de predicados que contidos nesses caminhos; essas informações estão disponíveis no Apêndice A.

Analizando-se o comportamento individual das rotinas do benchmark, através dos dados do Apêndice A e das Tabelas 3.3 e 3.4, nota-se que na maioria delas é maior o número de caminhos com o número de predicados que entre 4 e 7. Além disso, apenas 5 rotinas ---TRANSLIT, MAKEPAT, SUBST, DODASH, GETCMD --- contribuem com caminhos com o número de predicados que maior ou igual a 12. Essas rotinas podem ser consideradas atípicas, pois as três primeiras possuem muitos potenciais-du-caminhos e uma relação muito alta de caminhos não executáveis, e as duas últimas não apresentam nenhum caminho não executável. Outras rotinas com características atípicas são ARCHIVE, que não tem caminhos não executáveis, e EDIT e GETONE, que possuem também um grande número de potenciais-du-caminhos e uma porcentagem muita alta de caminhos não executáveis. Algumas dessas rotinas foram caracterizadas em [MAL91a] como os pontos mais influentes na determinação dos modelos para estimar as variáveis resposta número de casos de teste e número de caminhos não executáveis; particularmente, as rotinas TRANSLIT, MAKEPAT e SUBST foram as que demandaram maior número de casos de teste para satisfazer os critérios Potenciais Usos.

Para estudar o relacionamento entre o número de predicados de um potencial-du-caminho e sua executabilidade, assim como Malevris, explorou-se a 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 $q \geq 1$ --- aplicando-se um teste χ^2 e utilizando-se os dados das Tabelas 3.3 e 3.4. Foram obtidos $\chi^2 = 323.70$ e $\chi^2 = 304.63$, respectivamente, para os conjuntos TOTAL e TOTAL-R, com $v = 17$ (número de graus de liberdade) para os dois casos. Como $\chi^2_{(0.005)} = 35.718$ 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 cada valor de q e que deve existir alguma forma de dependência entre a executabilidade de um caminho e o número de predicados q nele contido.

A análise dessa dependência, ou melhor, a obtenção de um modelo para caracterizar essa associação, foi conduzida a partir dos dois conjuntos de programas considerados. Primeiramente, com q no intervalo de 1 a 18 (o número de potenciais-du-caminhos com $q=0$ foi somado ao número de caminhos com $q=1$) e, posteriormente, restringindo-se q ao intervalo de 1 a 12, uma vez que poucas rotinas contribuem para $q \geq 12$ (não foram excluídas as contribuições das rotinas que contribuem com $q \geq 12$ no intervalo $1 \leq q < 12$).

Foram explorados modelos polinomiais e modelos exponenciais, a exemplo de Malevris [MALE90]; 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 $1 \leq q \leq 9$, 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 dele ser executável, é também válida para potenciais-du-caminhos, no benchmark considerado. Os modelos exponenciais obtidos foram $p = 1.018 e^{-0.149q}$, com $r^2 = 95.0$, e $p = 1.015 e^{-0.153q}$, com $r^2 = 95.7$, respectivamente, para os conjuntos TOTAL-R e TOTAL. Esses modelos para esses conjuntos são apresentados detalhadamente no Apêndice A. Desta forma tem-se que esses modelos exponenciais expressam o relacionamento entre os valores apresentados e mostram comprovando a hipótese de

Malevris, que quando aumenta o número de predicados o número de caminhos executáveis tende a diminuir.

Considerando-se $1 \leq q \leq 18$ observou-se um comportamento quadrático tanto para o modelo exponencial quanto para o modelo polinomial; deve ser ressaltado que somente cinco rotinas "atípicas" contribuem para $12 \leq q \leq 18$. Para se extrair um resultado mais conclusivo no intervalo $1 \leq q \leq 18$ mais dados devem ser coletados e analisados, evitando-se que a análise fique influenciada pelo comportamento de uma rotina em particular; por exemplo, em um dos modelos obtidos nesse intervalo, um dos pontos de maior influência era $q = 17$ — a única rotina que contribui para este ponto é a rotina TRANSLIT que possui 5 caminhos executáveis e 5 caminhos não executáveis com $q = 17$; mais ainda, são apenas 10 dos 333 caminhos da rotina em questão.

Adicionalmente, foram explorados vários modelos excluindo-se completamente as cinco unidades com $q \geq 12$ dos dois conjuntos iniciais TOTAL e TOTAL-R. Os números de predicados encontrados para os potenciais-du-caminhos, para os dois conjuntos estão nas Tabelas 3.6 e 3.7. Obtiveram-se modelos semelhantes aos modelos obtidos considerando-se a contribuição dessas rotinas no intervalo $1 \leq q < 12$; os melhores modelos obtidos foram $p = 0.97 e^{-0.164q}$, com $r^2 = 95.6$, e $p = 0.965 e^{-0.160q}$, com $r^2 = 94.4$, apresentados com maiores detalhes no Apêndice A. Este fato mostra que não existe uma influência significativa dessas rotinas com $q \geq 12$ na determinação dos modelos para o intervalo $1 \leq q < 12$; e ainda evidencia, que para o intervalo $q \geq 12$, outro conjunto de rotinas deve ser caracterizado para viabilizar uma análise mais significativa para $q \geq 12$.

Um ponto a ser salientado é que foram considerados nesta análise todos os potenciais-du-caminhos exigidos pelo critério. Análise semelhante poderia ter sido realizada para conjuntos de caminhos completos que incluíssem todos os potenciais-du-caminhos requeridos. Mas estimando-se o número de potenciais-du-caminhos não executáveis, pode-se estimar o número de caminhos completos não executáveis, pois os

potenciais-du-caminhos não executáveis constituem padrões não executáveis, e qualquer caminho completo que os inclua será também não executável; consequentemente, a probabilidade desses caminhos completos serem não executáveis será maior que ou igual à probabilidade de qualquer potencial-du-caminho não executável nele incluído.

Tabela 3.6: Número de Predicados Contidos nos Potenciais-du-caminhos das Rotinas do Benchmark com $q \geq 12$

	0	1	2	3	4	5	6	7	8	9	10	11	12	tot
nex	0	19	41	44	85	100	145	175	83	55	30	42	0	819
exc	19	80	86	76	94	83	83	57	30	18	16	0	0	642
tot	19	99	127	120	179	183	228	232	113	73	46	42	0	1461

Tabela 3.7: Número de Predicados Contidos nos Potenciais-du-caminhos das Rotinas do Benchmark com $q \geq 12$, Excluindo-se as Unidades Recursivas

	0	1	2	3	4	5	6	7	8	9	10	11	12	tot
nex	0	19	40	37	78	91	140	156	57	49	22	36	0	725
exc	19	78	78	74	87	76	78	52	25	15	16	0	0	598
tot	19	97	118	111	165	167	218	208	82	64	38	36	0	1323

3.5 Considerações Finais

Neste capítulo apresentaram-se os principais resultados relativos à condução de um "benchmark", utilizando-se a ferramenta de teste POKE-TOOL. Procurou-se realizar estudos a exemplo dos principais trabalhos existentes na literatura, apresentados no capítulo anterior. Entre estes trabalhos estão os de caracterização de não executabilidade, realizados por Hedley e Hennel e de previsão de não executabilidade, realizados por Malevris.

Primeiramente, apresentou-se uma síntese de como foi conduzido o benchmark e quais foram os principais dados coletados durante a condução do benchmark. Em seguida, foram apresentadas, divididas em categorias, as principais causas de não executabilidade encontradas nos caminhos das rotinas analisadas; deve-se notar que muitas dessas causas podem ser eliminadas reescrevendo-se os programas. Caracterizar os caminhos não executáveis foi muito importante para poder, dado um programa, dizer qual a probabilidade dele conter muitos caminhos não executáveis, para poder orientar a obtenção de programas com um número reduzido de caminhos não executáveis, e para determinar heurísticas e facilidades para tratamento de não executabilidade.

A dificuldade encontrada para avaliar predicados compostos e determinar manualmente a executabilidade de caminhos com um grande número de predicados, foi uma motivação para realizar estudos para comprovar a hipótese de Malevris --- da influência do número de predicados de um caminho na sua executabilidade --- no benchmark e para estudar a influência de outras características do programa no número de caminhos não executáveis.

Foram obtidos vários modelos de estimativa bastante razoáveis para estimar a variável resposta: número de caminhos não executáveis, notando-se que várias características do programa, tais como: número de comandos de decisão, número de variáveis, número de nós com definição de variáveis, e número de potenciais-du-caminhos, influenciam no número de

caminhos não executáveis do programa.

Foram explorados vários modelos com o objetivo de validar as afirmações de Malevris. Pôde-se validar essas afirmações para os potenciais-du-caminhos no benchmark considerado, com o número de predicados do caminho q , no intervalo $1 \leq q \leq 9$, excluindo-se ou não as unidades recursivas. Deve-se ressaltar a inexistência de rotinas em número suficiente (somente 5) com $q \geq 12$, para se obter resultados mais conclusivos para o intervalo $1 \leq q \leq 18$.

De maneira geral, pode-se dizer que entre as principais causas de não executabilidade, encontram-se os laços e flags e que, em muitos desses casos, a heurística de Frankl, apresentada no Capítulo 2, pode ser aplicada. Outras facilidades que podem ser extraídas dessa experiência e que devem ser automatizadas, são: eliminação de padrões de não executabilidade, utilização de inconsistência entre predicados para determinar caminhos não executáveis, otimizações para a heurística de Frankl, etc. A descrição detalhada e os aspectos que envolvem a implementação dessas facilidades estão no capítulo seguinte.

CAPÍTULO 4

DETERMINAÇÃO DE EXECUTABILIDADE NAS ATIVIDADES DE TESTE

O objetivo principal de implementar facilidades para tratamento de caminhos não executáveis é auxiliar o usuário na geração de casos de teste para cobrir os elementos requeridos por um determinado critério, em particular os caminhos e associações exigidos pelos critérios Potenciais Usos, mas ainda não executados. E assim, poder:

- evitar que esforço e tempo sejam gastos na tentativa de gerar dados de teste para caminhos e associações não executáveis.
- auxiliar a geração de dados de teste para associações e caminhos executáveis.
- eliminar mais rapidamente caminhos, associações e padrões não executáveis.

Durante a condução do "benchmark", verificou-se a necessidade e importância de incorporar à POKE-TOOL certas facilidades para tratamento de não executabilidade de caminhos. Muitas das facilidades implementadas foram estabelecidas 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. Nesse sentido a heurística proposta por Frankl, mostrou-se bastante interessante, pois permite analisar em um caminho um laço em particular.

A ferramenta POKE-TOOL, na sua versão atual, pode ser utilizada para auxiliar tanto a seleção de casos de teste como para avaliar a adequação de um conjunto de casos de teste em relação aos critérios

Potenciais Usos; ela produz um conjunto de caminhos (ou associações) requeridos. Infelizmente, não é possível implementar uma ferramenta para determinar todos os caminhos e associações requeridas que não são executáveis pois, como foi dito anteriormente, a questão de determinar se um caminho ou associação é ou não executável é indecível.

Como é impossível determinar a executabilidade em todos os casos, optou-se por implementar um sistema interativo onde o usuário seleciona a associação ou caminho com o qual ele está trabalhando. O sistema apresenta todas as informações disponíveis e, para os casos em que nada pode ser decidido, o usuário poderá determinar ou ajudar o sistema a decidir sobre a executabilidade do caminho.

Como principais facilidades a serem implementadas, pode-se destacar:

- determinar associações não executáveis através de heurísticas; aplicação da heurística proposta por Frankl.
- determinar caminhos não executáveis obtendo-se inconsistências de predicados.
- eliminar associações não executáveis do arquivo de associações requeridas e eliminar os caminhos que possivelmente cobririam essa associação, do arquivo de potenciais du-caminhos requeridos.
- eliminar potenciais du-caminhos não executáveis por conterem inconsistências de predicados ou um padrão de não executabilidade.
- apresentar os predicados associados aos nós de um caminho dado.
- calcular o número de predicados de um caminho, para determinar a probabilidade do caminho ser executável.
- gerar os possíveis potenciais du-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.

De maneira resumida, para determinar e eliminar caminhos não executáveis, deve-se realizar as seguintes tarefas, que podem ser vistas como essenciais: aplicação de heurísticas, determinação e eliminação de caminhos que contenham inconsistências ou padrões, executar simbolicamente comandos e avaliar predicados associados aos nós. Na Seção 4.1 apresentam-se os aspectos principais que foram considerados para implementar essas tarefas. Na Seção 4.2 é dada uma breve descrição funcional da arquitetura da POKE-TOOL. Na Seção 4.3 são descritas as principais funções que foram incorporadas à POKE-TOOL para implementar as tarefas acima.

4.1 Aspectos Teóricos da Implementação

4.1.1 Aplicação de Heurísticas.

Para determinar não executabilidade utiliza-se a heurística proposta por Frankl, apresentada no Capítulo 2, e extensões dessa heurística, obtidas durante a condução do "benchmark" e que serão descritas nessa seção.

A heurística proposta por Frankl tem como objetivo determinar associações não executáveis utilizando-se análise de fluxo de dados e técnicas de avaliação simbólica. A idéia é eliminar do conjunto de caminhos candidatos a cobrir a associação, aqueles que não são viáveis ou por redefinirem variáveis da associação ou por não serem executáveis. Existem dois problemas básicos para sua implementação: como representar os candidatos e como avaliar os predicados dos laços.

Para Frankl os candidatos a cobrir uma associação $[i,(j,k),x]$ são caminhos completos que passam pelo nó i e alcançam o arco (j,k) , representados por expressões regulares [FRA87]. Para eliminar os candidatos que não são viáveis, numa primeira etapa, eliminam-se os que não são livres de definição.

Os caminhos candidatos a cobrir uma associação e que são livres de definição c.r.a x são dados pelo conjunto de caminhos representados pela extensão a ciclo para a associação $(i,(j,k),x)$. Utilizando-se o grafo(i), pode-se gerar os potenciais du-caminhos $\pi = (n_1, \dots, n_j, n_k)$ que dão origem à extensão a ciclo para a associação $[i,(j,k),x]$, e que contêm apenas candidatos viáveis, ou seja, livres de definição c.r.a x . Isso é facilitado pelo conjunto $deff(k)$ de cada nó k do grafo(i).

Ao conjunto gerado $(\pi, [i, (j, k), x])$, pode-se aplicar a segunda etapa proposta por Frankl: utilizar informação semântica para eliminar os candidatos. Serão eliminados aqueles potenciais du-caminhos estendidos a ciclo $(\pi, [i, (j, k), x])$, tal que $\pi = (n_1, \dots, n_j, n_k)$ e que para algum nó n_p $i <= p <= j$, $Pred(n_p)$ não é satisfeita, ou seja, $f_aval(n_p, \pi) = false$, e não existe possibilidade de redefinir $Var_Pred(n_p)$ em nenhum dos ciclos de L_p . Se todos os potenciais du-caminhos que representam extensões a ciclo para a associação puderem ser eliminados, a associação será não executável.

No exemplo da Figura 4.1, o nó i possui uma definição de i e gera o grafo(i) apresentado na Figura 4.2, onde a associação $[i, (4, 9), \{concorre, concurso, finalist, i\}]$ será requerida.

O conjunto de caminhos candidatos a cobrir essa associação requerida são os caminhos de i para $(4, 9)$, que são livres de definição c.r.a $v \in V = \{concorre, concurso, finalist, i\}$. Esses caminhos são representados por dois conjuntos estendido a ciclo: $(\pi_1, [i, (4, 9), V])$ e $(\pi_2, [i, (4, 9), V])$, onde $\pi_1 = (1, 2, 4, 9)$ e $\pi_2 = (1, 3, 4, 9)$. Aplicando-se a heurística a π_1 , tem-se que $Pred(4) = (i <= finalist)$ e $f_aval(4, \pi_1) = false$, $L_4 = \{(4)\}$, pois para $\lambda_4 = (4, 5, 6, 8, 4)$ e $\lambda_4 = (4, 5, 7, 8, 4)$, i é

redefinida. Percorrendo $\lambda_4 = \{4\}$ a variável i não é redefinida; portanto, o predicado não poderá ser satisfeito e o conjunto estendido a ciclo $(\pi_1, [1, (4,9), V])$ poderá ser eliminado. Analogamente, pode-se eliminar o conjunto $(\pi_2, [1, (4,9), V])$ e a associação é considerada não executável.

No exemplo acima, a análise dos caminhos através do laço do nó 4 não precisaria ser realizada pois todas as variáveis do predicado associado ao nó 4, $\text{Var_Pred}(4) = \{i, \text{finalist}\}$, são também variáveis da associação analisada, e seria impossível redefinir alguma das variáveis do predicado por caminhos livres de definição c.r.às variáveis da associação. Essa situação leva a definição da Extensão 1 da heurística de Frankl definida abaixo.

Extensão 1 da Heurística: Seja $(\pi, [i, (j,k), V])$ um conjunto estendido a ciclo para a associação $[i, (j,k), V]$ e $\pi = (n_1, \dots, n_j, n_k)$ um potencial du-caminho $\in \text{grafo}(i)$. Seja n_p , $1 \leq p \leq j$, um nó que possui um predicado tal que $f_{\text{aval}}(n_p, \pi) = \text{false}$. Se $\text{Var_Pred}(n_p) \subseteq V$ então $L_p = \{ (n_p) \}$ e o conjunto estendido a ciclo $(\pi, [i, (j,k), V])$ pode ser eliminado.

Existem alguns casos nos quais a determinação de caminhos não executáveis pode ser realizada mesmo sem a avaliação de predicados. Utilizando-se o programa da Figura 4.3, para analisar a associação $<7, (4,9), \{i\}>$ requerida pelo grafo 7, eliminando-se os caminhos que não são livres de definição com respeito a variável i, tem-se $L_4 = \{ (4) \}$. Nesse caso não há necessidade de avaliar o predicado do nó 4; o caminho $\pi = (7, 8, 4, 9)$, que gera o conjunto estendido a ciclo $(\pi, [7, (4,9), \{i\}])$, pode ser eliminado e a associação $<7, (4,9), \{i\}>$ ser considerada não executável, pois esse é o único conjunto estendido a ciclo para essa associação. Isso porque a condição do nó 4 é verdadeira, pois nenhum caminho de 4 para 7 e novamente de 7 para 4 redefine uma das variáveis associadas ao predicado do nó 4. Isso não é válido para o exemplo da Figura 4.4, pois para a associação $<6, (2,8), \{i\}>$ existem caminhos de 2 para 6 que redefinem $\{\text{done}\}$. Dessa maneira pode-se fazer a seguinte consideração (Extensão 2 da heurística de Frankl).

```

void write_finalist(concorre,concurso)
char *concorre[];
int concurso;
{
    int i, finalist;
    i
    finalist
    concorre
    concurso
1   i = 1;
1   finalist = 10;
1   if (concurso == 3)
2       premiado = 0;
3   else
3       premiado = 5;
4   while (i ≤ finalist)
5   {
5       if (i≤ premiado)
6           printf("premiados %s", concorre[i]);
7       else
7           printf("%s", concorre[i]);
8       i = i +1;
8   }
9 }

```

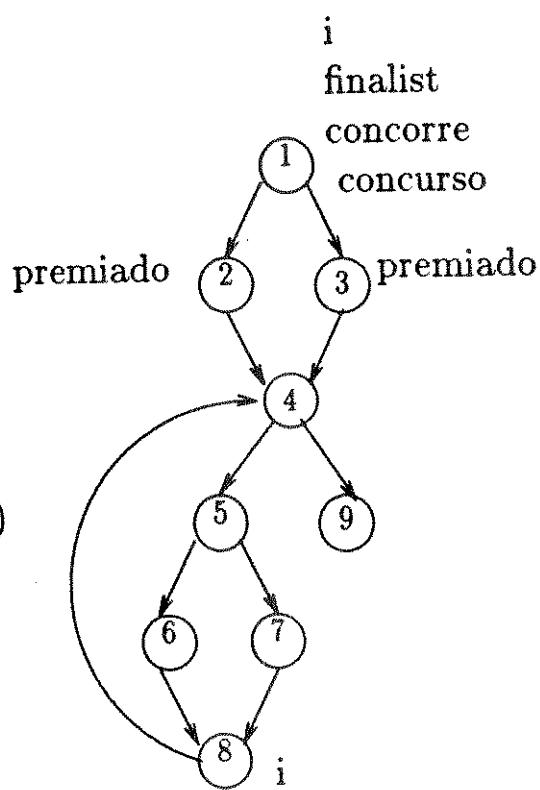
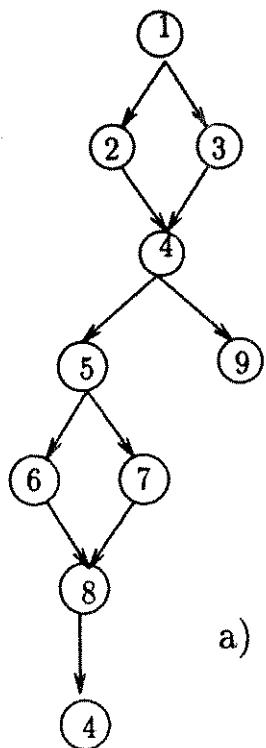
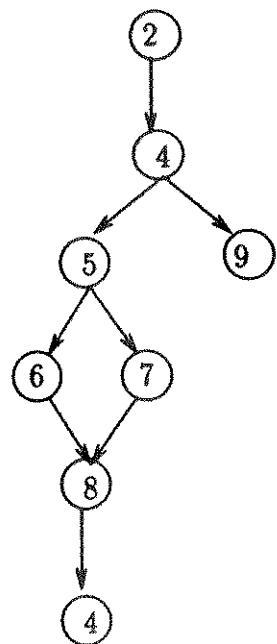


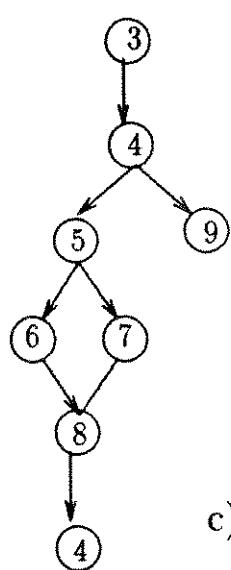
Figura 4.1: Exemplo de Utilização da Heurística de Frankl.



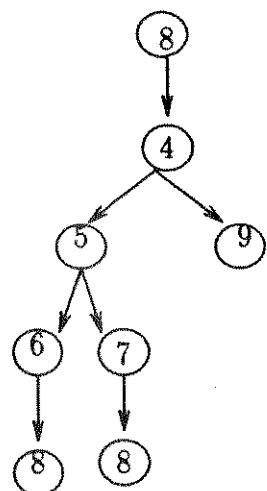
a)



b)



c)



d)

Figura 4.2: Grafos(i) Gerados pelo Programa da Figura 4.1.

Extensão 2 da heurística: Seja uma associação $[i,(j,k),V]$, $N_p = \{ n_p \in \text{grafo}(i) : n_p \text{ é um nó cabeça de laço} \}$. Se existir um nó $n_p \in N_p$, tal que, todo caminho $\sigma = (n_p, \dots, n_m, \dots, n_l)$ (caminhos de n_p para n_l , tal que $i \neq p$ e $m \neq p$) é livre de definição c.r.a $\text{Var_Pred}(n_p)$ e $f_{\text{aval}}(n_p, \sigma) = \text{true}$; e se qualquer caminho $\gamma = (n_l, \dots, n_p)$, (caminhos de n_l para n_p), livre de definição c.r.a $v \in V$ é também livre de definição c.r.a $\text{Var_Pred}(n_p)$, então para o caminho $\pi = (n_l, \dots, n_j, n_k)$ com $i < p \leq j$, que gera um conjunto estendido a ciclo $(\pi, [i, (j, k), V])$, tem-se: $f_{\text{aval}}(n_p, \pi) = \text{true}$ se $\text{Pred}(n_p)$ precisar valer true no caminho π , ou $f_{\text{aval}}(n_p, \pi) = \text{false}$, caso contrário. Se $f_{\text{aval}}(n_p, \pi) = \text{false}$, o conjunto $(\pi, [i, (j, k), x])$ poderá ser eliminado. Observe que se não existir um nó $n_p \in N_p$ que satisfaça as condições acima, o conjunto $(\pi, [i, (j, k), x])$ poderá ser executável.

Por conveniência técnica, para simplificar a utilização de expressões de caminhos e para representar os candidatos a cobrir a associação, Frankl assumiu que os programas aos quais o algoritmo proposto para implementar a heurística é aplicado não possuem comandos *repeat-until*. Utilizando-se os conjuntos estendidos a ciclo para representar os candidatos a cobrir a associação e os grafos(i) para eliminar os que não são livres de definição c.r. às variáveis do predicado, o mesmo tratamento pode ser dado para os comandos *repeat-until*, *for* ou *while*.

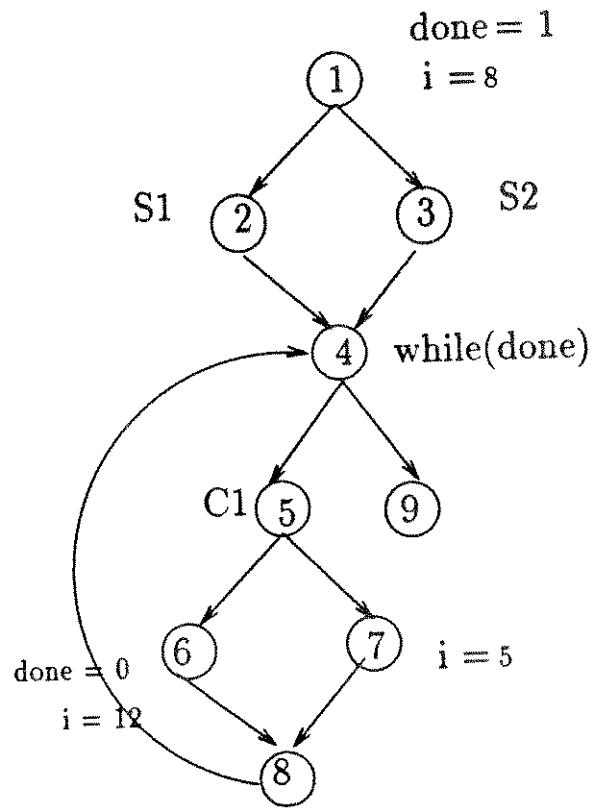


Figura 4.3: Exemplo para a Extensão 2 da Heurística de Frankl

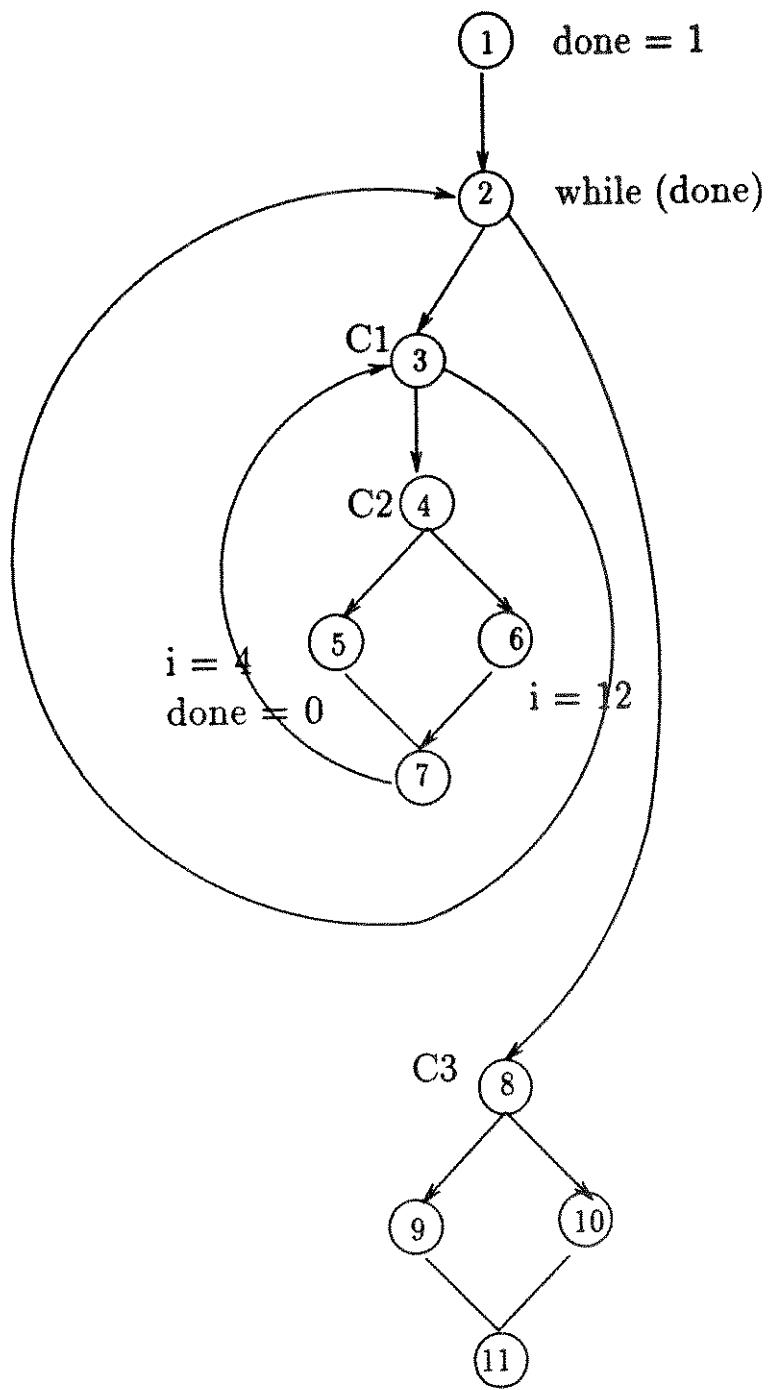


Figura 4.4: Contra-exemplo para a Extensão 2 da Heurística de Frankl

4.1.2 Padrões de não executabilidade.

Seja um caminho $\pi = (n_1, \dots, n_m)$, $n_1 = s$ e k um nó do caminho π tal que $1 \leq k < m$, sendo que $\text{Pred}(n_k)$ não é satisfeita, ou seja, $f_{\text{aval}}(n_k, \pi) = \text{false}$; o caminho π pode ser considerado não executável. Qualquer caminho que contiver um subcaminho não executável, no caso π , será não executável. A sequência de nós (n_1, \dots, n_m) é não executável e pode ser considerada um padrão de não executabilidade. Se $n_1 \neq s$ e $\text{Pred}(n_p)$ não é satisfeita independentemente do caminho de s para n_1 , a consideração acima permanece válida.

No exemplo da Figura 4.4 a sequência (ou subcaminho) $\pi = (1, 2, 8)$ é não executável pois $f_{\text{aval}}(2, \pi) = \text{false}$, ou seja o predicado associado ao nó 2 não é satisfeito. Os caminhos $\sigma = (1, 2, 8, 9, 11)$ e $\gamma = (1, 2, 8, 10, 11)$ podem ser eliminados pelo padrão 1 2 8.

Padrões de não executabilidade podem ser determinados durante a aplicação de heurísticas para determinar associações não executáveis. Se π , dado por $\pi = (n_1, \dots, n_j, n_k)$, é tal que $(\pi, [i, (j, k), V]) \in \Delta$ extensão a ciclo para uma associação $[i, (j, k), V]$ e π foi desconsiderado, é porque o potencial du-caminho $\pi = (n_1, \dots, n_j, n_k)$ é não executável. No exemplo da seção anterior, Figura 4.3, obtém-se o padrão 1 2 4 9 do conjunto estendido a ciclo $\pi = (1, 2, 4, 9)$ que foi desconsiderado para cobrir a associação $<1, (4, 9), \{\text{done}\}>$, por não redefinir as variáveis do predicado do nó 4. Se uma associação é não executável todos os conjuntos estendidos a ciclo foram desconsiderados e, portanto, todos os potenciais du-caminhos que a cobrem, ou seja, de i para (j, k) , são não executáveis. Ainda pelo mesmo exemplo, o padrão $1 N_i^* 4 9$, onde $N_i = \{n : n \text{ não redefine } (\text{done}, i)\}$ é obtido da associação não executável $<1, (4, 9), \{\text{done}, i\}>$ e elimina os potenciais du-caminhos $\beta = (1, 2, 4, 9)$ e $\gamma = (1, 3, 4, 9)$.

4.1.3 Execução Simbólica

Técnicas de execução simbólica têm como objetivo derivar expressões algébricas que representem a execução de um caminho ou conjunto de caminhos de um programa, derivando uma condição para que o caminho seja executado ou uma expressão simbólica que represente as variáveis de saída (resultado da computação do caminho) em termos das variáveis de entrada [FRA87].

A computação de um caminho pode ser obtida em termos da computação individual dos nós que compõem o caminho. A partir de um conjunto inicial de valores para as variáveis do programa executam-se simbolicamente os comandos associados ao primeiro nó do caminho e produz-se um novo conjunto de valores para o próximo nó do caminho e assim sucessivamente.

A execução simbólica de um comando consiste basicamente em avaliar simbolicamente o lado direito e atribuir à variável do lado esquerdo do comando o resultado simbólico obtido. A execução simbólica de um comando apresenta muitos problemas para ser implementada na prática [MAY90]. Entre eles:

- referências às variáveis compostas (vetores e matrizes) e ponteiros.
- chamadas de procedimentos e parâmetros passados por referência, efeitos colaterais, variáveis globais.
- variáveis que compartilham a mesma posição de memória ou variáveis que são sinônimos ("Aliasing").

A Figura 4.5 ilustra o problema com referência as variáveis compostas e ponteiros. Como avaliar a expressão $A[i] \leq A[j]$ se i e j possuem valores indeterminados; ou seja a quais elementos do vetor A a condição se refere. Uma alternativa apresentada é usar um pseudocaminho;

quando uma referência a um array é encontrada, é seguido um caminho através do programa para todas as possibilidades [HOW77]. Um conjunto finito de valores poderia ser atribuído às variáveis *i* e *j* do exemplo, mas determinar e testar todas as possibilidades quase sempre é impraticável, por limitações de tempo e de custo. A solução adotada foi considerar, seguindo o modelo de dados usado na POKE-TOOL [MAL91a,CHA91d], a definição dessas variáveis, e atribuir um valor simbólico indeterminado a cada um de seus elementos, já que não é possível em geral determinar que objeto de dado está sendo referenciado.

Outra complicação é introduzida pelas chamadas de procedimentos e funções. Pelo exemplo da Figura 4.6, na expressão *a=leitura(&np,&nl)*, *nl* e *np* são passados por referência. Como saber qual o valor atribuído às variáveis *a*, *np*, e *nl*. Representar numa expressão o resultado da execução de uma função ou procedimento não é uma tarefa trivial; isso equivaleria a realizar avaliação global, que possui inúmeras restrições de implementação [FRA87]. Escolher um caminho arbitrário através da função ou procedimento e executá-lo pode ocasionar inconsistências.

Ainda no exemplo da Figura 4.6, a função leitura lê um número *n*, retornando o valor 1, se *n* é ímpar e 0, caso contrário, os parâmetros *nimpares* e *npares* que foram passados por referência, podem ser alterados ou não. Isso deve ser levado em consideração para avaliar o predicado *if (a)*, e não é traduzido pela escolha arbitrária de um caminho através de leitura.

Como não é possível saber a priori que caminho será tomado dentro da função, é necessário empregar técnicas de avaliação simbólica global. Para simplificar a implementação, nessa fase inicial, optou-se em retornar, para chamadas de funções, um valor simbólico indeterminado, o mesmo acontecendo para as variáveis globais e variáveis passadas por referência, que podem ser alteradas pelas funções.

Variáveis que podem ser vistas como sinônimos são as que ocupam a mesma posição de memória. Isso pode ocorrer de várias maneiras num

programa; se x e y são parâmetros formais que podem ser alterados, ou são passados por referência, e um mesmo valor é passado para x e y na chamada do procedimento; igualmente, se uma variável global é passada como parâmetro por referência; se p e q são ponteiros para a mesma variável; se A é um vetor, A[i] e A[j] são sinônimos, se i = j; o uso de variáveis COMMON do FORTRAN é também um exemplo de variáveis que são sinônimos.

Para implementar execução simbólica de comandos, restringiu-se o tipo de "aliasing" que pode ocorrer nos programas testados. Considera-se que, quando o programa foi chamado, todos os parâmetros reais correspondentes aos parâmetros formais passados por referência são distintos entre si e distintos das variáveis globais que ocorrem no programa. Os valores atribuídos a arrays e ponteiros são valores simbólicos indeterminados e devem ser únicos. Essas restrições podem representar perda de informação útil na avaliação de predicados; talvez o estudo de técnicas heurísticas possa determinar quais dos "aliasing" potenciais realmente acontecem no programa.

Para realizar execução simbólica no ambiente POKE-TOOL, que é uma ferramenta que pode ser configurada para diversas linguagens, é desejável que a execução dos comandos e avaliação dos predicados possam ser independentes da linguagem de programação do programa em teste. Por isso os comandos, após serem reconhecidos numa análise sintática são representados numa forma padrão para que possam ser executados simbolicamente. Os valores para as variáveis do programa são atualizados num conjunto denominado computação para o nó.

```
scanf ("%d %d", &i, &j);
A[0] = 0;
A[1] = 5;
.....
A[10] = 9;
if (A[i] ≤ A[j]) .....
```

Fig 4.5 Problema com Arrays na Execução Simbólica

```

int leitura (int *npares , int *nimpares)
{
    int n;
    div_t k;
    scanf("%d",&n);
    k = div(n,2);
    if (k.rem == 0)
    {
        (*npares)++;
        return(1);
    }
    else
    {
        (*nimpares)++;
        return(0);
    }
}

```

```

{
    a = leitura(&np,&ni);
    if (a)
    {
        .....
    }
}

```

Fig 4.6 Problema com Chamada de Procedimentos e Funções na Execução Simbólica

4.1.4 Avaliação de predicados

Avaliar o predicado de um nó num conjunto estendido a ciclo, obtido a partir do grafo(i), e que represente os possíveis candidatos a cobrir uma associação $[i,(j,k),Var_Assoc]$, pode exigir a execução simbólica de um conjunto de caminhos. Podem existir muitos caminhos que vão do nó de entrada para o nó i ; neste caso, fica difícil obter o conjunto inicial de valores para as variáveis do programa associado ao nó i .

Frankl [FRA87] apresenta uma técnica para avaliar um conjunto de caminhos, chamada avaliação parcial de predicados. Mas numa primeira etapa, pretende-se que as rotinas implementadas executem apenas os comandos de um nó, ou um caminho simples de i para (j,k) . Os valores atribuídos às variáveis inicialmente serão valores simbólicos indeterminados, que serão transformados à medida que os comandos vão sendo executados simbolicamente.

No exemplo da Figura 4.1, para avaliar o predicado do nó 4 ($i \leq finalist$), no conjunto estendido a ciclo dado por $\pi = (1,2,4,9)$, basta executar os comandos dos nós 1 e 2, pois trata-se de um caminho simples. Para executar os comandos do nó 1, toma-se o conjunto $\{concorre1, concurso1, finalist1, i1, premiado1\}$ de valores simbólicos para as variáveis do programa. Após a execução dos comandos do nó 1, esse conjunto de valores foi transformado no conjunto $CS(1,(1,2,4,9)) = concorre1, concurso1, 10, premiado1$, que será utilizado para execução dos comandos do nó 2. Como resultado dessa execução é obtido o conjunto $CS(2,(1,2,4,9)) = \{concorre1, concurso1, 10, 1, 0\}$; com esse conjunto o predicado do nó 4 pode ser avaliado true, e $f_aval(4,\pi) = false$.

Além de existirem muitas restrições para executar simbolicamente os comandos do nó, nem sempre a avaliação dos predicados pode ocorrer de maneira direta; por isso a participação do usuário será decisiva, fornecendo informações sobre o programa em teste; essas informações nada mais são que fatos associados a um nó ou caminho, verdades que podem ser assumidas. Assim, o usuário poderá fornecer o conjunto de valores

inicial para as variáveis do programa em determinado nó, ou fornecer o resultado da avaliação do predicado para que o sistema determine um padrão não executável ou fornecer o próprio padrão. Uma das vantagens é que isso exige um bom conhecimento do programa que está sendo testado, e na tentativa de auxiliar o sistema na avaliação de predicados, algum erro poderá ser detectado no programa em teste.

No exemplo da Figura 4.1 tem-se que o caminho $\pi = (2,4,9)$ é requerido pelo grafo(2). Os valores das variáveis para o conjunto inicial, utilizado para a avaliação do nó 2, são valores indeterminados. Ao terminar a execução dos comandos do nó 2, $CS(2,\pi) = \{ \text{concorre1, concurs01, finalist1, i1, 0} \}$, com esses valores não é possível avaliar o predicado do nó 4. Nesse caso o usuário poderá auxiliar na avaliação, determinando qual o conjunto inicial de valores das variáveis para avaliar o nó 2.

No exemplo da Figura 4.7, para avaliar a associação $(1,(6,7),x)$, tem-se o potencial-du-caminho $\pi = (1,2,3,6,7)$, um dos candidatos que será analisado; para esse potencial-du-caminho tem-se $Pred(6) = (i > 0)$ e $f_aval(6,\pi) = \text{true}$, já que $f=4$ e $i \geq f$. O usuário pode determinar isso facilmente e ajudar o sistema fornecendo o resultado da avaliação, $f_aval(6,\pi) = \text{true}$. Ou ainda, fornecer o valor de i no nó 6 pois o laço que começa e termina no nó 3, será executado exatamente f vezes e, ao ser encerrado, $i=f=4$.

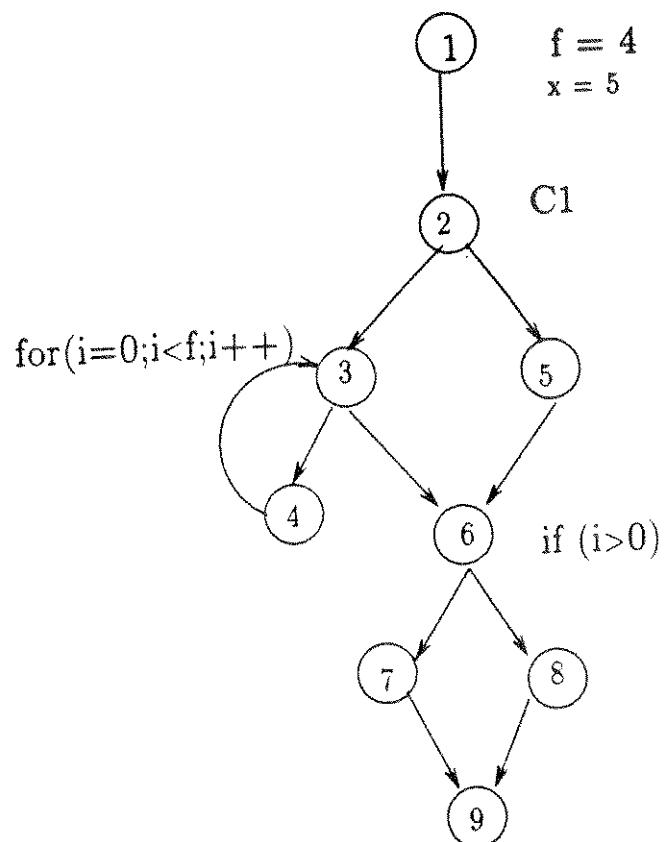


Fig 4.7 Exemplo de Introdução de Fatos pelo Usuário
na Avaliação de Predicados

4.2 Aspectos da Arquitetura e da Implementação da POKE-TOOL

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, e permite que o usuário a configure para a linguagem de programação de seu interesse. Nessa seção é apresentada uma síntese dos aspectos de sua arquitetura, extraída de [CHA91d,MAL91a].

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.

As principais funções da POKE-TOOL podem ser vistas na Figura 4.8. De forma resumida, podemos dizer que a POKE-TOOL determina o grafo de fluxo de controle do programa a ser testado (Geração do Grafo do Fluxo de Controle). Esse grafo é estendido incorporando-se informações de fluxo de dados, obtendo-se o grafo-def (Extensão do Grafo de Fluxo de Controle); isso é feito associando-se a cada nó i , do grafo de fluxo de controle, o conjunto de variáveis definidas no nó i , de acordo com um modelo de dados pré-estabelecido. No contexto de teste de software, para estender o grafo de fluxo de controle utilizam-se três tipos de ocorrências de variáveis: definição, uso e indefinição. O uso de uma variável, que ocorre quando seu valor é acessado na posição de memória correspondente, do ponto de vista dos critérios Potenciais Usos, não precisa ser identificado.

Abaixo descrevem-se os pontos mais relevantes do Modelo de Dados proposto para a POKE-TOOL [MAL91a,CHA91d].

A passagem de valores entre procedimentos através da passagem de parâmetros pode ser por: valor, referência ou por nome [GHE87]. De

acordo com o modelo de dados adotado pela POKE-TOOL [CHA91d,MAL91a], se a passagem for por referência ou por nome, considera-se que a variável é um parâmetro de saída. As definições decorrentes de possíveis definições em chamadas de procedimentos são distinguidas das demais e são ditas definidas por referência; esta distinção é utilizada na geração dos grafos(1), descrita posteriormente, ou seja, na determinação dos caminhos livres de definição para as variáveis definidas em 1. Resumidamente, uma variável v_i definida por referência faz parte do conjunto de variáveis definidas em 1, ou seja $v_i \in \text{defg}(1)$, para construção dos grafo(1), mas qualquer definição por referência de v_i em um nó $j \neq i$ não é considerada como redefinição de v_i ; este enfoque é conservador no sentido de que não deixa de requerer nenhum fluxo de dados que possa existir, ou seja, a seleção de caminhos e associações é mais rigorosa.

Uma complicação é introduzida quanto ao tratamento de ponteiros e de variáveis compostas: - variáveis estruturadas (vetores e matrizes) e registros -, pois não é possível, em geral, determinar-se estaticamente o elemento particular destas variáveis que está sendo referenciado. Para variáveis estruturadas adotou-se que a definição de um elemento da variável implica a definição da variável. Para variáveis do tipo registro adotou-se a mesma abordagem, ou seja, qualquer definição de um campo de um registro é considerado definição do registro. Variáveis do tipo ponteiro são tratadas exatamente como uma variável comum e não é feito o tratamento de definições por dereferencião. Outra vez, não foram tratados esses tipos de definição porque é, em geral, impossível saber estaticamente que objeto dado estaria sendo definido por dereferencião. Observe-se que o enfoque adotado para variáveis compostas não é conservador, pois alguns fluxos de dados existentes podem não ser requeridos.

No modelo de fluxo de dados adotado, considera-se que no nó de entrada ocorre uma definição dos parâmetros e das variáveis globais que ocorrem na unidade em teste.

Ocorre uma indefinição de variável se sua localização não estiver amarrada na memória ou se seu valor não for acessível. A indefinição pode ocorrer devido ao encerramento da execução da unidade; neste sentido o nó de saída tem uma indefinição de todas as variáveis locais.

Um outro conceito importante [MAL91a] é o conceito de arco primitivo: baseia-se no fato de existirem arcos dentro de um grafo de fluxo de controle que são sempre executados quando um outro arco é executado; esses arcos são ditos não essenciais para a análise de cobertura. Um arco que sempre é executado quando um outro arco é executado é denotado por arco herdeiro. Se todo caminho completo que inclui o arco a sempre incluir o arco b , b é arco herdeiro de a e a é arco ancestral de b . O arco primitivo é todo arco que não é herdeiro de nenhum outro. A função Cálculo dos Arcos Primitivos calcula os arcos primitivos que serão posteriormente utilizados para descrição dos caminhos e associações requeridos pelos Critérios Potencias Usos.

A Instrumentação auxilia na determinação dos caminhos efetivamente executados pelo conjunto de casos de teste fornecido. Os descritores são utilizados para verificar se o critério foi satisfeito. A partir do grafo def e do conjunto de arcos primitivos constróem-se os grafos(i) e ainda são fornecidos os descritores de caminhos e associações em termos dos arcos primitivos a serem utilizados na avaliação de um conjunto de casos de teste qualquer (Avaliação).

O grafo(i) é obtido a partir do grafo def e fornece todos os caminhos livres de definição c.r.a. qualquer variável definida em i para todo nó e todo arco alcançável a partir do nó i . A proposição é a de se construir um único grafo(i) para cada nó i tal que $\text{defg}(i) \neq \emptyset$, obtendo-se, por construção, uma minimização de caminhos e associações requeridos. Na realidade, os grafo(i) incluem somente os potenciais-du-caminhos a partir do nó i .

O algoritmo para obtenção dos grafo(i) é apresentado em [MAL89a, CHA91d]. Esse algoritmo consiste essencialmente de uma busca em

profundidade no grafo def a partir de nós que tenham definições de variáveis. Um nó k pertencerá a um grafo(i) se existir pelo menos um caminho livre de definição do nó l para o nó k c.r.a pelo menos uma das variáveis definidas em l . Associa-se a cada nó k do grafo(i) um conjunto de variáveis denominado $deff(k) \subseteq defg(i)$. Para qualquer variável $v \in deff(k)$, qualquer caminho $\pi = (l, \dots, k)$ no grafo(i), é livre de definição c.r.a v . Um nó k pode dar origem a q imagens distintas no grafo(i). Denota-se por $f_{\pi q}(k)$ a q -ésima imagem de k . É importante salientar que considerando-se dois caminhos distintos π_1 e π_2 , quaisquer duas imagens distintas no grafo(i), $f_{i\pi_1}(n) \neq f_{i\pi_2}(n)$, de um nó $n \in N$, $deff(f_{i\pi_1}(n)) \neq deff(f_{i\pi_2}(n))$, $i = 1, 2$.

Dado um grafo(i) com os seus respectivos arcos primitivos, a automatização dos Critérios Potenciais Usos é realizada com pequenas variações dos descritores dos caminhos e associações requeridos por esses critérios.

Na hipótese da POKE-TOOL ser utilizada na avaliação da adequação de um conjunto de casos de teste, são determinados os caminhos (associações) efetivamente executados e verifica-se se os aceitadores correspondentes aos descritores dos caminhos (associações) requeridos estão no estado final (o critério foi satisfeito); caso contrário, é fornecida ao usuário uma lista de caminhos requeridos pelo critério e não executados pelo conjunto de casos de teste. No caso da POKE-TOOL ser utilizada para auxiliar na geração de casos de teste, o conjunto "default" de caminhos requeridos pelo critério (obtidos durante a construção dos grafo(i)) é fornecido.

Entre esses caminhos (ou associações) requeridos pode existir um elemento não executável. Não estava previsto para esta versão da POKE-TOOL a determinação de caminhos e associações não executáveis. Na próxima seção serão apresentadas funções que foram incorporadas na POKE-TOOL para tratamento de não executabilidade.

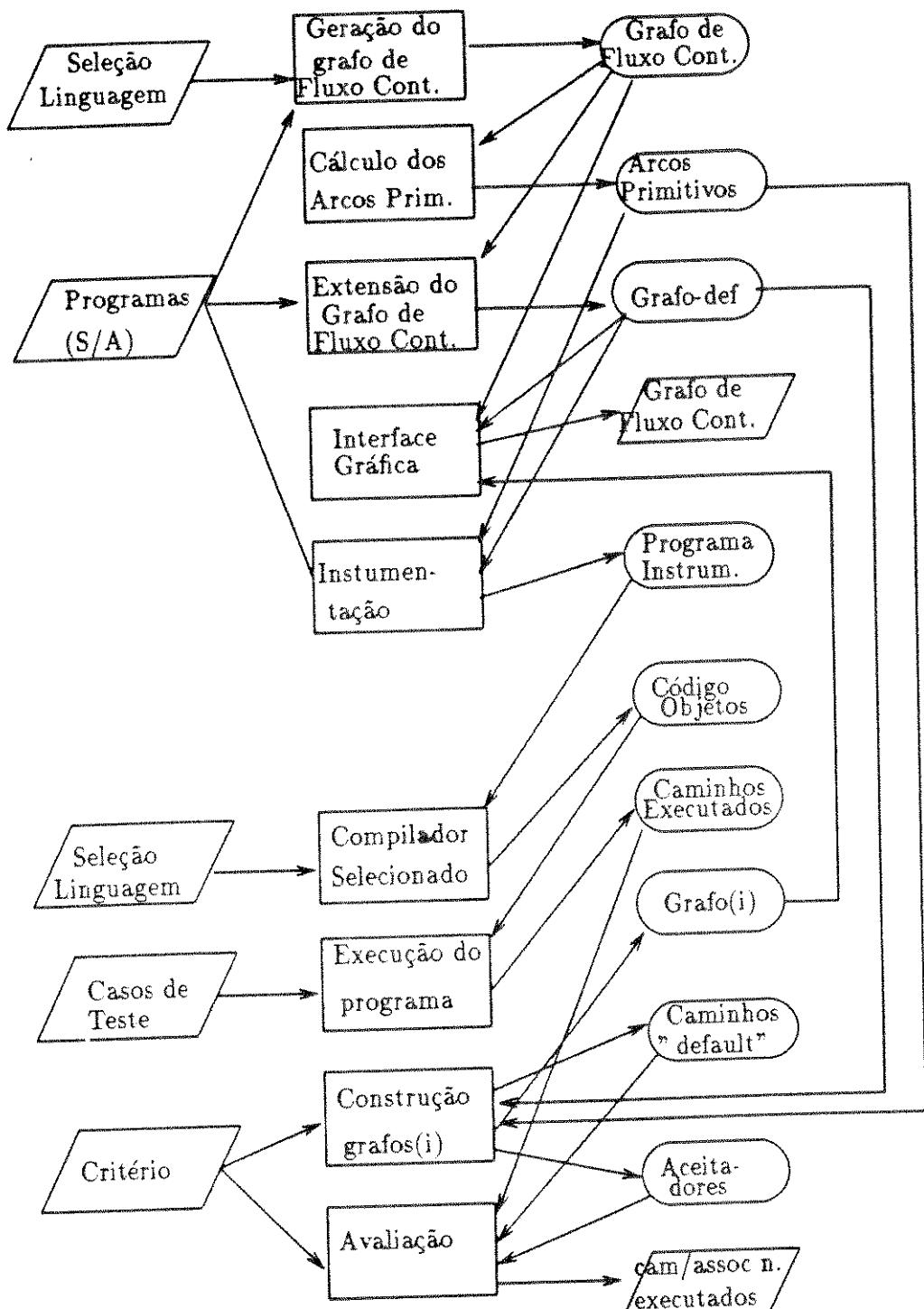


Figura 4.8: Arquitetura da Ferramenta POKE-TOOL

4.3 Incorporação de Facilidades à POKE-TOOL para Tratamento de Não Executabilidade

Para implementar as facilidades para tratamento de não executabilidade descritas, segundo as considerações apresentadas, serão incorporadas à POKE-TOOL as seguintes funções, ilustradas nas Figuras 4.9 e 4.10:

Geração do Grafo-Def/Pred: estender o grafo-def, associando-se a cada nó i os predicados e o número de predicados do nó.

Geração de Dados para Execução Simbólica: consiste em representar todos os comandos associados ao nó i de forma que eles possam ser executados simbolicamente. Os comandos serão representados numa expressão cujos elementos serão identificados atribuindo um tipo a cada elemento, que pode ser: variável, constante, função, operadores, símbolos como ')', '(', '[', ']', etc.

Execução Simbólica: a partir de um conjunto inicial de valores para as variáveis do programa, computação simbólica inicial para o nó i, executar simbolicamente os comandos no caminho π de i para j, produzindo um novo conjunto de valores, computação simbólica para o nó j, $CS(j, \pi)$ representando como as variáveis foram transformadas no caminho π de i para j.

Avaliação de Predicados: Avaliar os predicados de um nó, utilizando-se do conjunto de valores produzidos pela execução simbólica e das informações fornecidas pelo usuário tais como, avaliação de predicados, valores iniciais para variáveis indeterminadas, fatos que podem ser associados a cada caminho ou nó que esteja sendo avaliado.

Determinação de Extensões a Ciclo para uma Associação: os conjuntos estendidos a ciclo para uma associação representam os vários caminhos que podem cobrir a associação, ou seja, caminhos livres de definição c.r.às variáveis da associação, e são obtidos do grafo(i) correspondente.

Eliminação de Caminhos ou Associações não Executáveis: Uma vez determinado a não executabilidade de uma associação ou caminho, o elemento não executável deverá ser eliminado do arquivo de caminhos ou associações requeridos. Ao receber um novo padrão, via aplicação de heurísticas ou via usuário, todos os potenciais du-caminhos que contiverem este padrão deverão ser eliminados do arquivo de potenciais du-caminhos requeridos.

Cálculo do número de predicados do caminho: Para um potencial du-caminho dado, pode-se calcular, a partir do grafo-def/pred o número de predicados do caminho; esse módulo é utilizado para avaliar a influência do número de predicados na executabilidade do caminho.

Determinação de Caminhos Não Executáveis: dado um potencial du-caminho, a determinar sua executabilidade. Um potencial du-caminho será não executável se contiver um padrão não executável ou se a avaliação de seus predicados não for consistente, nesse caso gera-se um novo padrão não executável.

Determinação de Associações Não Executáveis: dada uma associação e os conjuntos estendido a ciclo que a cobrem, determinar sua executabilidade. Para realizar essa tarefa está prevista a implementação da heurística proposta por Frankl [FRA88].

Resumidamente, dada uma potencial-associação serão determinados, utilizando-se o grafo(i) e o grafo-def/pred, os conjuntos estendido a ciclo, representando os possíveis candidatos a cobrirem a associação dada. Para determinar se a associação é não executável, será aplicada a heurística proposta por Frankl e extensões a cada conjunto estendido a ciclo, onde serão analisados os predicados de cada laço, e verificado através dos grafos(i) 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 potenciais du-caminhos que a cobrem, constituindo-se um novo padrão de não executabilidade. A executabilidade de caminhos será 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 poderão ser fornecidos pelo usuário, quando serão eliminados os potenciais-du-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.

A incorporação dessas facilidades à POKE-TOOL tem como objetivo principal auxiliar a geração de casos de teste para satisfazer os critérios suportados pela POKE-TOOL. Para isso está prevista uma parte automática onde as funções são realizadas sem interação com o usuário. Nesta etapa, são determinados e eliminados o maior número possível de caminhos e associações. Na outra etapa, que envolve interação com o usuário, este poderá auxiliar a determinar não executabilidade, ajudando, por exemplo, a avaliar predicados. Alternativamente o sistema poderá ajudar o usuário a determinar dados de teste para associações e caminhos executáveis apresentando maneiras para cobrir a associação e como os predicados devem ser satisfeitos ao longo do caminho. Ainda nessa etapa, a eliminação de caminhos, associações e padrões não executáveis determinados pelo usuário, pode ser realizada.

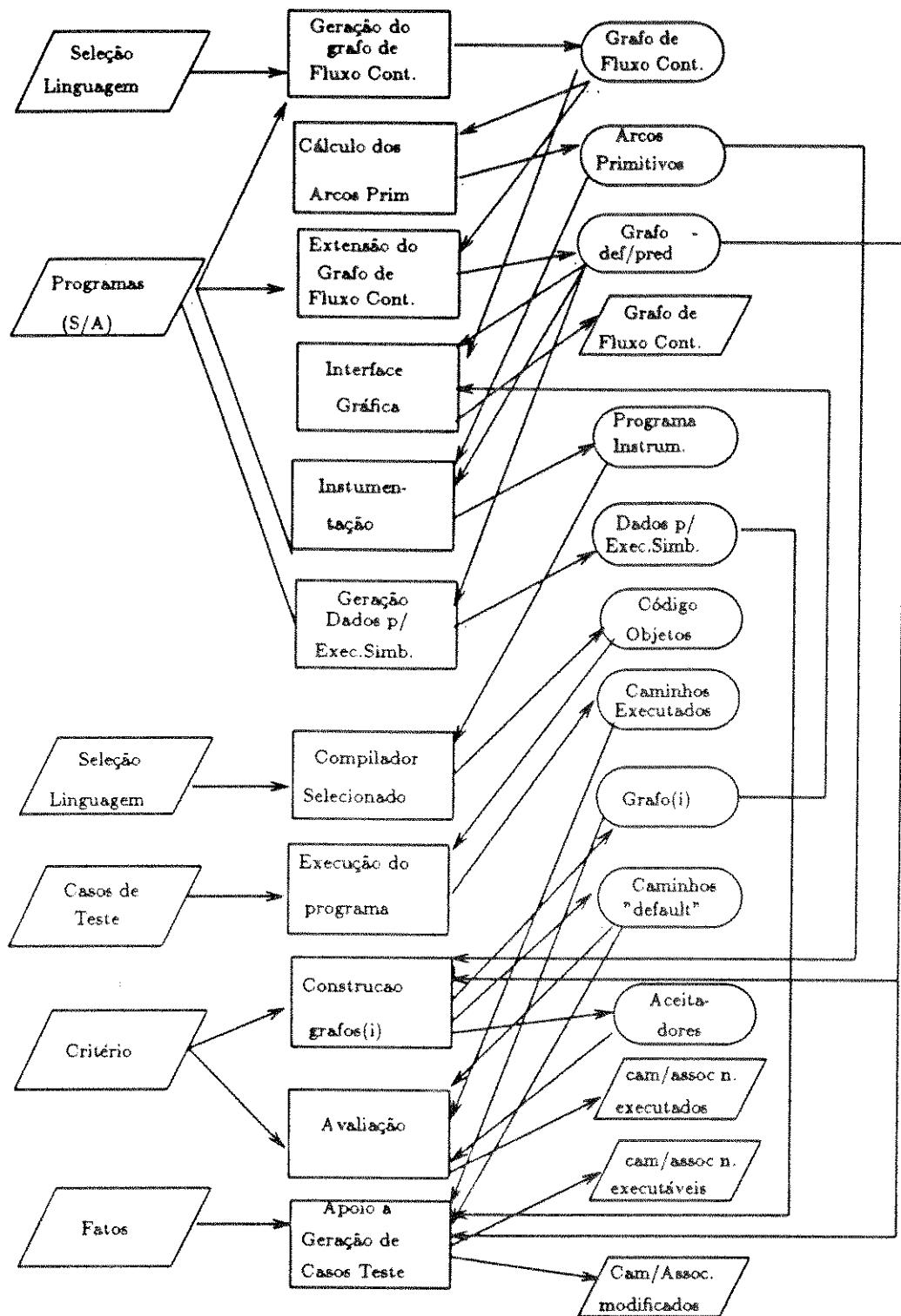


Figura 4.9: Introdução de Novas Funções na POKE-TOOL

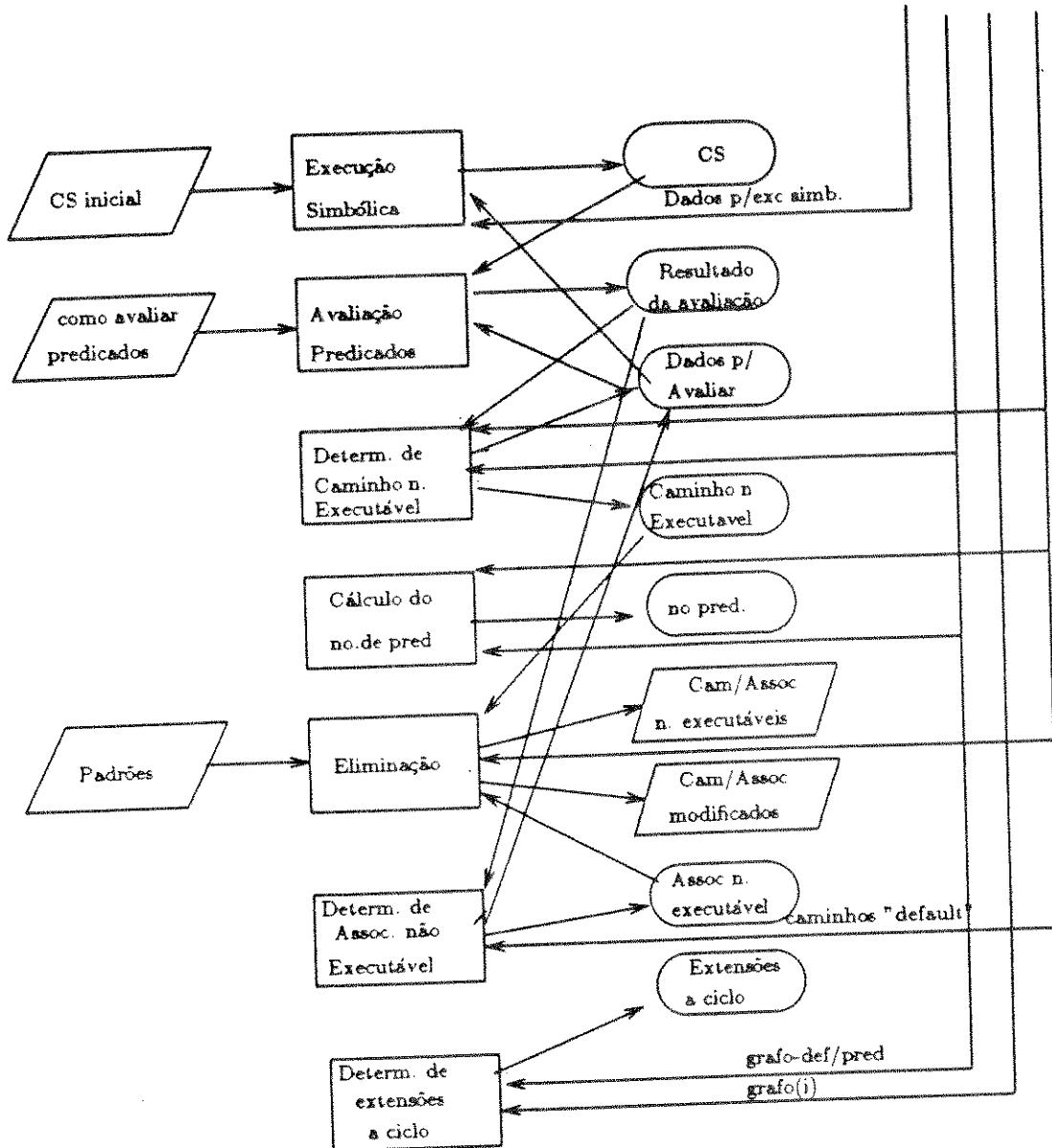


Figura 4.10: Detalhamento da Função Apoio à Geração de Casos de Teste

4.4 Considerações Finais

Neste capítulo foram apresentadas as principais facilidades que foram incorporadas à POKE-TOOL. De maneira geral, o objetivo é facilitar a geração de casos de teste para caminhos e associações requeridos, eliminando padrões de não executabilidade automaticamente, implementando heurísticas, verificando inconsistências de predicados.

Também foram discutidos como implementar e quais os aspectos relevantes e dificuldades de implementação dessas facilidades, entre eles: como representar candidatos a cobrir uma associação; como avaliar predicados e executar comandos simbolicamente; a utilização de padrões não executáveis, como otimizar a heurística proposta por Frankl; etc.

Propondo-se a utilização de extensões a ciclo para representar os caminhos candidatos e determinar essas extensões através do grafo(1), considera-se como um candidato os caminhos que já são livres de definição c.r. às variáveis da associação; assim não é necessário realizar o primeiro passo proposto por Frankl.

Duas extensões para a heurística de Frankl foram introduzidas. A extensão 1, facilita a aplicação da heurística em muitos casos, pois não é necessário entrar em detalhes sobre o laço que está sendo considerado. A extensão 2, permite determinar não executabilidade sem necessidade de avaliar o predicado de um laço.

Esses aspectos foram úteis para determinar manualmente as associações não executáveis e são úteis para simplificar e otimizar a implementação de heurísticas. A utilização de padrões permite que sejam eliminados muitos caminhos, cuja não executabilidade é originada pela mesma causa. É importante automatizar a eliminação de padrões, pois o número de caminhos que contém um padrão pode ser muito grande.

Como foi dito, a avaliação e execução simbólica de comandos apresenta inúmeras dificuldades de implementação. Nessa primeira etapa, optou-se por um sistema interativo, onde o usuário ajude a resolver esses problemas. A participação do usuário para avaliar predicados e introduzir fatos, é fundamental.

Neste capítulo também foi dada uma síntese de aspectos da ferramenta POKE-TOOL e das principais funções que consideram os aspectos discutidos e que foram acrescentadas a esta ferramenta, para tratar não executabilidade. O capítulo seguinte descreve os principais módulos que implementam essas funções, assim como um exemplo de utilização e avaliação desses módulos.

CAPÍTULO 5

ASPECTOS DE IMPLEMENTAÇÃO

Como foi dito anteriormente, a POKE-TOOL é uma ferramenta de testes que suporta a utilização dos Critérios Potenciais Usos, implementada em C, desenvolvida para o sistema operacional MS-DOS, para ambientes do tipo PC. A versão presente suporta programas escritos em C, mas já está sendo configurada para outras linguagens de programação, entre elas COBOL, FORTRAN, PASCAL.

A operação da POKE-TOOL realiza-se em duas fases: uma estática e a outra dinâmica. Na fase estática são produzidos o conjunto de caminhos requeridos pelo critério todos-potenciais-du-caminhos e o conjunto de associações requeridas pelos critérios todos-potenciais-usos e todos-potenciais-usos/du. Essas informações podem ser utilizadas para projeto de casos de teste ou na análise de adequação de conjuntos de casos de teste.

Na outra fase, a dinâmica, o usuário poderá executar seus casos de teste e avalia-los segundo um dos três critérios acima.

A POKE-TOOL é composta de vários módulos que se comunicam através de arquivos. Na Figura 5.1 é apresentado um diagrama contendo os módulos e os diversos produtos gerados. Nesta figura, os retângulos representam os módulos, os losângos representam as entradas fornecidas pelo usuário, e os círculos os produtos gerados; as linhas tracejadas o fluxo de controle e as linhas cheias o fluxo de informação.

O módulo Poketool é o responsável pela comunicação entre a ferramenta e o usuário e pela sequenciação das atividades de teste através da ativação dos demais módulos.

O módulo LI é sensível à linguagem na qual está escrita a unidade em teste, pois realiza a tradução dessa unidade para uma versão na linguagem intermediária (LI).

O módulo Chanomat gera o grafo de fluxo de controle (GFC) da unidade e uma nova versão da unidade em LI, onde cada comando está associado ao nó correspondente. Esses dois módulos implementam a função Grafo de Fluxo de Controle da POKETOOL.

O módulo Pokernel é o responsável pelo restante da análise estática da unidade em teste, gerando as informações estáticas adicionais necessárias ao teste dinâmico da unidade. O Pokernel implementa as seguintes funções da POKE-TOOL: Cálculo dos Arcos Primitivos, Extensão do Grafo de Fluxo de Controle, Instrumentação, Construção dos Grafos(i) e Geração de Descritores.

O módulo Gera Executável fornece as condições para a geração do programa executável da versão instrumentada e engloba a função Compilador Selecionado.

O módulo Executa Caso de Teste controla a execução dos casos de teste, salvando as entradas, a saída e o caminho executado para cada caso de teste; implementa a função Execução do Programa da POKE-TOOL.

Finalmente, o módulo Avaliador identifica os caminhos ou associações executados pelos casos de teste e fornece uma análise da cobertura do conjunto de casos de teste fornecido, implementando a função Avaliação da POKE-TOOL. Discussões mais detalhadas sobre a implementação e operação da POKE-TOOL são apresentadas em [CHA91a, CHA91b, CHA91c].

O objetivo da implementação de rotinas para serem incorporadas à POKE-TOOL é facilitar a tarefa do usuário para gerar casos de teste, permitindo a eliminação automática de caminhos ou associações não executáveis e/ou simplesmente apresentar como os predicados devem ser

satisfetos para que o usuário possa gerar seus casos de teste, ou determinar a não executabilidade interativamente.

No Capítulo 3 foram descritas as principais funções a serem incorporadas à POKE-TOOL para implementar as facilidades acima. Algumas dessas funções: Geração do Grafo Def/Pred, e Geração de Dados para Execução Simbólica, foram implementadas por rotinas que foram adicionadas ao ou fazem parte do módulo Pokernel, e geram, respectivamente o grafo-def/pred e os dados utilizados posteriormente para execução simbólica do programa. As demais funções foram implementadas pelo módulo Apelo à Geração de Casos de Teste. A interface da POKE-TOOL também foi alterada para suportar essas novas funções. Isso pode ser visto na Figura 5.2. A seguir, dá-se uma descrição de como foi realizada a implementação desse módulo e um exemplo de utilização dessas funções.

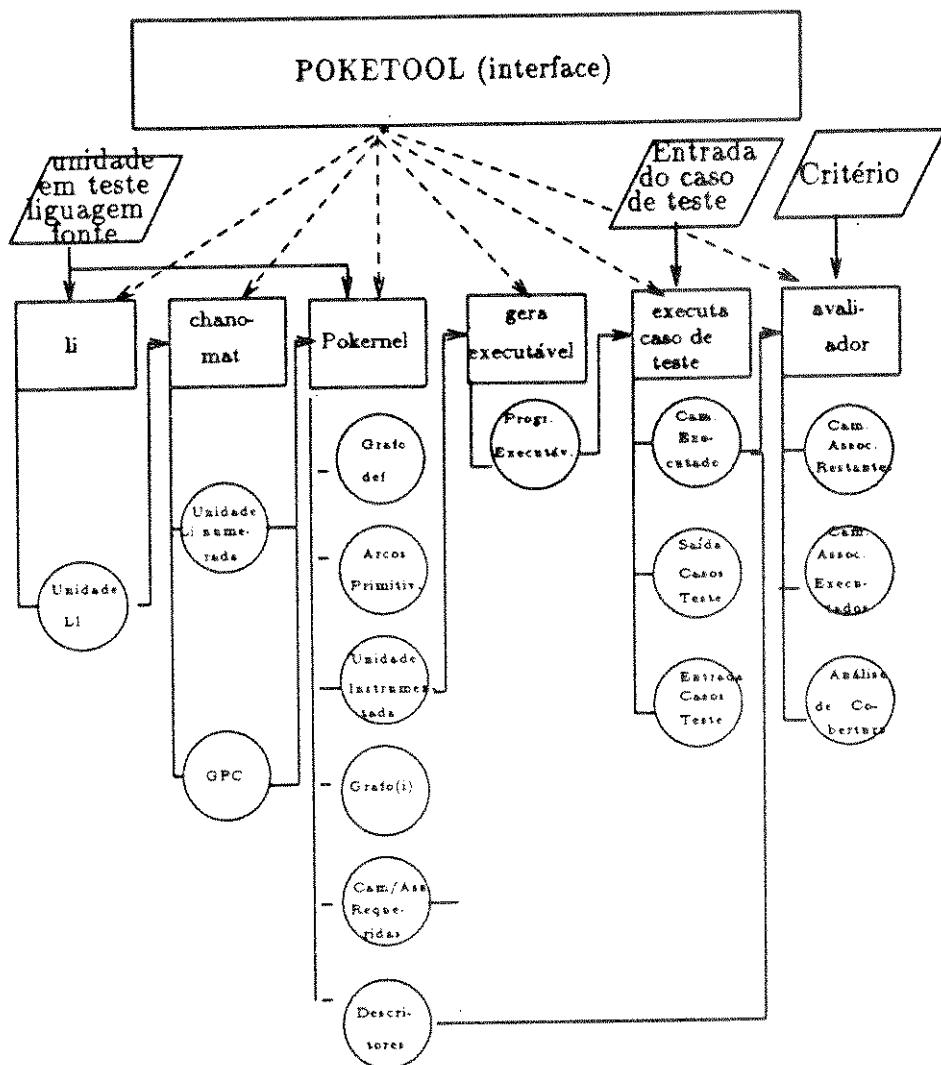


Figura 5.1: Projeto da POKE-TOOL

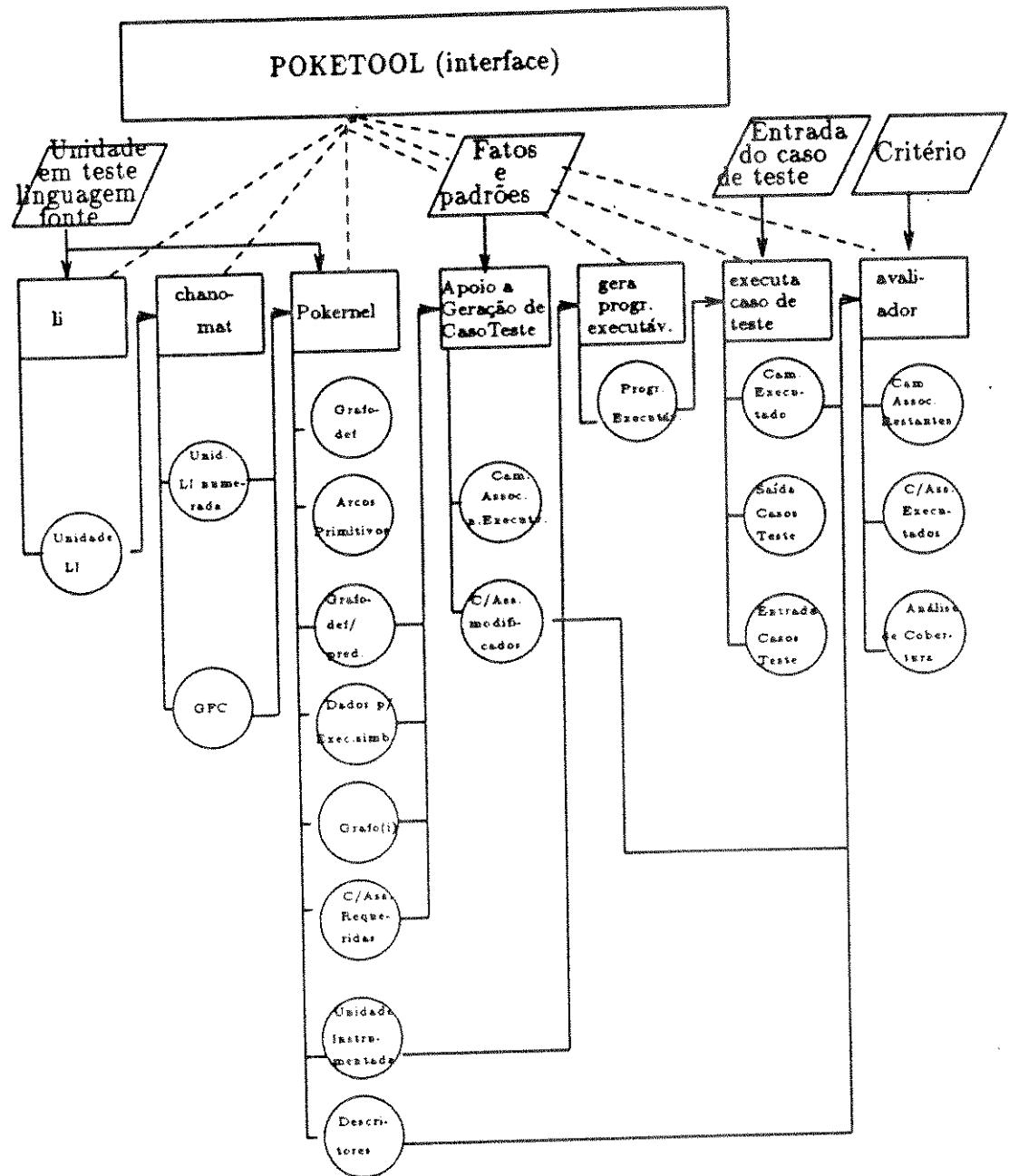


Figura 5.2 Projeto Modificado da POKE-TOOL

5.1 O Módulo Pokernel da POKE-TOOL.

Este módulo é o responsável por parte da análise estática do código fonte da unidade. Esta análise é fundamental para aplicação dos Critérios Potenciais Usos, bem como para gerar informação utilizada para executar os comandos e avaliar os predicados de um nó.

O módulo POKERNEL gera:

- (1) conjunto de arcos primitivos (arquivo ARCPREM.TES).
- (2) o grafo-def (GRAFODEF.TES).
- (3) a unidade instrumentada (arquivo TESTEPRO.C).
- (4) o conjunto de potenciais-du-caminhos requeridos pelo critério todos-potenciais-du-caminhos (arquivo PDUPATHS.TES).
- (5) as associações requeridas pelos critérios todos-potenciais-usos e todos-potenciais-usos/du (arquivo PUASSOC.TES).
- (6) os descritores para os Critérios Potenciais Usos (arquivo DES_PDU.TES, DES_PU.TES, DES_PUDU.TES).

Foram acrescentadas novas funções ao módulo POKERNEL para que sejam também gerados:

- (7) grafo-def/pred (arquivo GRAFOPRED.TES).
- (8) informações para realizar execução simbólica, representando o programa de uma maneira especial (arquivo PROGSIMB.TES).

O módulo Pokernel tem como entrada os arquivos UNIDADE.C (na versão atual da POKE-TOOL), UNIDADE.GFC e UNIDADE.NLI, onde UNIDADE é o nome da unidade em teste e os arquivos mencionados contêm, respectivamente, a unidade propriamente dita, o GFC e a versão da unidade em LI numerada. O Pokernel é também dividido em sub-módulos que realizam as funções descritas acima.

O sub-módulo Cal_prim calcula os arcos primitivos que descrevem os caminhos e associações exigidos pelos critérios Potenciais Usos. O sub-módulo Parserli gera o grafo-def e o grafo-def/pred, a unidade instrumentada e a unidade instrumentada numa forma que possa ser

executada simbolicamente. O sub-módulo Des-pot-du-cam determina os grafos(1) e através destes, os caminhos e associações requeridos e os descritores utilizados para avaliar a adequação de conjuntos de casos de teste em relação aos critérios Potenciais Usos.

O módulo Parsell realiza uma análise sintática e semântica do código fonte para gerar os grafo-def e a unidade instrumentada. Também foram acrescentadas novas funções para gerar a extensão grafo-def/pred, e uma unidade instrumentada para execução simbólica de comandos. Essa análise é simplificada porque a unidade em teste possui uma representação especial que é o programa em LI.

Utilizando-se a LI a análise é simples porque cada declaração, comando sequencial, ou expressão condicional é representado por um átomo da LI, onde esse átomo possui ponteiros para o código fonte e um número indica o seu nó no GFC. Dessa maneira, basta tomar os ponteiros para o código fonte e analisar o trecho da unidade, verificando a presença de definições, predicados (o número de predicados de um comando condicional também é oferecido pela LI) associados a cada nó, armazenando cada comando numa forma em que possa ser executado simbolicamente, reconhecendo os comandos que alteram o fluxo de controle e inserindo pontas de prova para a instrumentação do programa.

A base para realização dessas tarefas, que são realizadas concomitantemente, é um analisador sintático descendente da LI, específico para a linguagem do programa, que vai reconhecendo os comandos e tomando as ações necessárias.

A seguir, apresenta-se uma síntese da geração dos grafo-def e grafo-def/pred e das principais estruturas de dados utilizadas para representá-los.

1) Extensão do grafo de fluxo de controle

Para estender o grafo de fluxo de controle, os analisadores léxico e sintático, identificam as variáveis e os predicados associados a cada nó. Para isso, eles possuem rotinas que manipulam as estruturas de dados que representam esses grafos. Essas rotinas seguem o modelo de dados (conjunto de construções que causam a definição de variáveis) apresentados no capítulo anterior e detalhados em [CHA91d], para a versão atual da POKE-TOOL.

O grafo-def/pred é mantido através de uma estrutura de dados responsável pelo GFC e uma estrutura de dados chamada *infono*. Para obter as informações do nó *i* do GFC, deve-se consultar o *i*-ésimo elemento de *infono*. Entre as informações dos elementos de *infono* para o grafo-def encontram-se três conjuntos: o conjunto de variáveis definidas (*defg*), o conjunto de variáveis possivelmente definidas por referência (*def_ref*) e o conjunto de variáveis indefinidas (*undef*) no nó. Esses conjuntos são representados através de bits (onde cada bit corresponde a uma variável). Inicialmente esses conjuntos são ajustados com o valor 0 e existe um número de identificação para cada variável que serve para identificar o bit associado aos conjuntos *defg*, *def_ref* e *undef*.

O grafo-def é estendido e torna-se grafo-def/pred pela introdução de outras informações na estrutura *infono* referentes aos predicados dos nós: um campo *num_pred* que é um inteiro que contém o número de predicados do nó; se *num_pred* > 0, então o nó possui predicado e o campo *pred* contém o predicado propriamente dito.

VARIAVEIS DEFINIDAS nos NOS do modulo cap4.c

Variaveis Definidas = Vars Defs
Variaveis possivelmente definidas por Referencia = Vars Refs
Predicado associado ao no = Pred

NO' 1
Vars defs: concorre concurso i finalist
Vars refs:
Pred: (concurso==3)
NO' 2
Vars defs: premiado
Vars refs:
Pred:
NO' 3
Vars defs: premiado
Vars refs:
Pred:
NO' 4
Vars defs:
Vars refs:
Pred: (i<=finalist)
NO' 5
Vars defs:
Vars refs:
Pred: (i<=premiado)
NO' 6
Vars defs:
Vars refs:
Pred:
NO' 7
Vars defs:
Vars refs:
Pred:
NO' 8
Vars defs: i
Vars refs:
Pred:
NO' 9
Vars defs:
Pred:

Figura 5.3: Exemplo de um Grafo-def/pred para o Programa da Figura 4.1 do Capítulo 4.

2) Instrumentação e Geração de informação para a execução simbólica de comandos

A instrumentação é a atividade que gera uma nova versão da unidade em teste. Essa versão é denominada unidade instrumentada e possui pontas de prova (instruções de escrita) que geram um "trace" do caminho executado.

Durante a análise sintática da LI, pode-se saber onde inserir as pontas de prova e ter-se acesso ao código fonte da unidade através de ponteiros que a LI possui. Além de inserir uma ponta de prova, também deve ser gerada uma nova unidade, para isso existe um procedimento que copia o código fonte associado ao comando reconhecido (através dos ponteiros da LI).

Ao analisar um comando da LI que pode ser um "statement" ou uma condição, ele é armazenado no arquivo PROGSIMB.TES, numa forma que possa ser executado simbolicamente, juntamente com o código correspondente e o nó associado.

Cada comando é representado por uma expressão dada por um ponteiro PTREXP para uma fila. Cada elemento da fila representa um termo reconhecido da expressão, é do tipo ELEM_EXPR e contém três campos: 1) tipo, inteiro que indica o tipo do termo reconhecido na expressão; pode representar uma variável (número de identificação da variável), operador binário, operador unário, operador de atribuição, funções, constantes (inteiros, real, etc..), e símbolos especiais como '[', ']', '(', ')'; 2) info, array de chars que contém o termo propriamente dito; 3) next, ponteiro para o próximo elemento da expressão. Se o termo for do tipo função ele será seguido dos parâmetros passados por referência que poderão ser alterados e de uma marca indicando fim de parâmetros possivelmente redefinidos; essa marca consiste em atribuir ao campo info o inteiro FIMPARAM.

A expressão reconhecida será armazenada na forma pós-fixada, para que a execução simbólica seja mais facilmente realizada. Para isso, está disponível uma tabela de operadores que contém entre outras informações, precedência de cada operador. Também existe uma estrutura de dados chamada **Infonesimb** que possui dois campos **stat** e **cond**, que são ponteiros para o arquivo **PROGSIMB.TES** para que se possa recuperar os comandos associados a cada no'("statement" e condição).

comando **k = 9;**

posfix	dado simb tipo	significado info	significado tipo
k	2	k	no. ident. k
9	60	9	cte. inteira
=	63	=	opr. atrib.

Figura 5.4: Representação de um Comando Instrumentado para Execução Simbólica.

5.2 O Módulo de Apoio à Geração de Casos de Teste

Este módulo auxilia a geração de casos de teste, pois permite determinar e eliminar automaticamente e/ou interativamente caminhos não executáveis. Evita, dessa forma, a tentativa de gerar casos de teste para algumas associações e caminhos não executáveis; ou, se não for possível determinar sua executabilidade, mostra para uma associação as possíveis maneiras em que ela poderá ser executada e, para um dado caminho como os predicados associados devem ser satisfeitos.

A parte automática realiza-se logo após a execução do módulo **Pokernel**. Tem como entrada os arquivos **GRAFOPRED.TES**, **PDUPATHS.TES**, **PUASSOC.TES**, **PROGSIMB.TES**. Os arquivos **PUASSOC.TES** e **PDUPATHS.TES** serão alterados pois para cada caminho ou associação não executável que foi detectado, será colocada uma marca (*). As associações serão eliminadas através da heurística de Frankl e otimizações dadas pelas extensões. Os caminhos serão eliminados através de inconsistências entre os seus predicados. Além disso, outros dois arquivos serão gerados: **PDUNEXEC.TES** E **PUNEXEC.TES**, contendo, respectivamente, os caminhos e associações não executáveis.

A parte interativa será ativada se o usuário selecionar um dado caminho ou associação que não possua marca de não executabilidade. Para um dado caminho os predicados serão exibidos e o usuário poderá decidir sua executabilidade, avaliando predicados ou introduzindo dados para auxiliar uma nova avaliação pelo sistema, o mesmo acontecendo para determinar as associações. O usuário poderá fornecer associações ou caminhos não executáveis que serão eliminados automaticamente pelo sistema, ou ainda fornecer um padrão não executável; neste caso, serão eliminados os caminhos que contiverem o padrão fornecido. Um exemplo da interação com o usuário é dado no final deste capítulo.

O módulo de apoio à geração de casos de teste pode ser dividido em sub-módulos que implementam as funções descritas no Capítulo 3. Esses sub-módulos são descritos a seguir.

1) Execução Simbólica

Definiu-se anteriormente a execução simbólica de caminhos executando-se os comandos associados a cada nó em termos de um conjunto inicial representando os valores das variáveis do programa.

Considerando as restrições para se implementar execução simbólica, o módulo implementado executa comandos simples; o resultado da execução é dado pelo vetor COMP_SIMB. COMP_SIMB representa o estado corrente das variáveis do programa. Quando uma definição de variável ocorre, COMP_SIMB[num_var], onde num_var é o número de identificação da variável, é atualizado. Quando uma variável é referenciada, seu valor é recuperado de COMP_SIMB. Cada elemento de COMP_SIMB é do tipo VALORSIMBOLICO e possui um campo inteiro tipo indicando os possíveis tipos resultantes de operações em C; os tipos considerados são apenas os tipos correspondentes às variáveis simples e o tipo indeterminado (para tipos correspondentes às variáveis compostas que não serão consideradas: arrays, ponteiros,...); e também possui um outro campo valor representando o valor simbólico da variável que varia conforme o tipo especificado.

Os comandos correspondentes a um nó i são armazenados no arquivo PROGSIMB.TES e são recuperados pelos ponteiros de infonosimb[i]. O algoritmo de execução simbólica é dado abaixo. Cada elemento da expressão (do tipo ELEMEXP) é analisado. Se tipo não é de um operador, seu valor simbólico é obtido e colocado numa pilha dada por topo_res. Se o elemento é um operador, os operandos necessários são retirados da pilha e a operação, segundo os tipos dos operandos obtidos é realizada; o resultado da operação é novamente colocado na pilha. O resultado final desta pilha é retornado pelo algoritmo dado a seguir.

```

VALORSIMBOLICO exec_simb (PTREXP *posf_expr)
/* posf_expr aponta para a fila que contem a expressao a ser
executada */
{
    while get_elem_expr(&posf_expr->inicio, &tipo, info)
    {
        if é operador
            push_res(avalia_operador(tipo,info),&topo_res);
        senao
            push_res(valor_simbolico(posf_expr,tipo,info),&topo_res);
        }
    return(pop_res(&topo_res));
}

```

int get_elem_expr(expr,tipo,info): Obtém o primeiro elemento da expressão apontada por expr. Tipo e info são parâmetros de saída que contêm o elemento obtido. Retorna 0 se expr é nulo, indicando que a expressão terminou.

VALORSIMBOLICO valor_simbolico (expr,tipo,info): Devolve a o valor simbólico do operando dado por tipo e info. O valor simbólico dependerá do tipo do operando, que determinará os campos da estrutura VALORSIMBOLICO. Assim, se o tipo de operando é uma constante inteira, a estrutura devolvida será dada por tipo = CTE_INT, e valor igual ao inteiro que representa a constante. Se tipo é uma função, o tipo retornado será INDET; além disso, COMP_SIMB[var] , para cada var, tal que var é uma variável encontrada em expr até FIM_PARAM, e que possivelmente foi redefinida pela função, ou uma variável global, deverá receber o valor indeterminado.

VALORSIMBOLICO avalia_opr(tipo,info): Conforme o tipo do operador, dado pelo parâmetro tipo, serão retirados da pilha dada por topo_res, os operadores necessários. Segundo os tipos dos operandos obtidos e segundo uma tabela que possui a semântica para os operadores, uma operação é

selecionada e uma rotina correspondente é executada. Se um operador não foi encontrado na tabela, um resultado indeterminado é retornado para a operação. Se nenhuma operação foi encontrada para o operador e tipos de operandos dados, a variável global `sit_erro` receberá valor 1, para indicar erro na execução simbólica.

2) Avaliação de Predicados

A avaliação de predicados dá-se de duas maneiras. Uma delas é sem interação com o usuário. Nesse caso executam-se os comandos do nó i, inclusive o comando condicional. Se foi possível avaliar o predíco, o resultado é comparado com o valor esperado; o resultado da comparação é retornado. A outra maneira é realizar avaliação interagindo com o usuário, se não foi possível avaliar o predíco. O usuário poderá entrar valores para as variáveis e uma nova avaliação é realizada, ou ainda, ele próprio determina a avaliação do predíco.

3) Módulo `det_assoc_nexec`

Esse módulo implementa as funções `determina_extesões-a-ciclo`, `determina-associação-não-executável` e parte da função `eliminação`, descritas no Capítulo 4.

Tem como entrada uma associação dada pelo arco do tipo PAIRINT, um vetor de bits `var_assoc` que contém as variáveis da associação, um ponteiro para o grafo(i) correspondente, cuja estrutura e algoritmo de geração são apresentados em [CHA91a]. A estrutura de dados utilizada para implementar os grafos(i) é uma lista ligada, onde cada elemento dessa lista é um nó do grafo(i). Cada elemento k dessa lista contém os campos: 1) `infosuc`, aponta para uma outra lista ligada contendo os próximos nós do grafo(i). `Infosuc` é do tipo INFODFNO que contém os campos `num_no_G` (número do nó no GFC) e `deff(k)` (conjunto de variáveis definidas no nó i, mas não redefinidas no caminho de i para k); 2)

repetido, indica se k tem o mesmo número no GFC que outro nó do grafo(i); 3) **sucgrfi**, aponta para lista de sucessores do nó. É do tipo SUCGRAFOI que possui os campos **num_grafo_i**, o campo **tipo** (indica se o arco formado pelo nó k e seu sucessor é primitivo ou herdeiro), o campo **no_address** (endereço do nó sucessor na lista de nós do grafo(i)), e o campo **next** (próximo nó sucessor).

O primeiro passo é percorrer o grafo(i), gerando os conjuntos estendidos a ciclo que pertencem à extensão a ciclo para a associação. Para isso existe um algoritmo que realiza uma busca em profundidade no grafo(i). A vantagem de utilizar os grafos(i) é que eles oferecem todos os possíveis potenciais du-caminhos a partir do nó i, que poderão ser eliminados facilmente se não alcançarem o arco da associação e se possuirem redefinição de variáveis de var_assoc (dada pelo deff do nó).

Existe uma variável **conj_est_ciclo** que é um vetor de no máximo MAXCONJCICLO elementos, sendo que **conj_est_ciclo[i]** é um ponteiro para uma lista ligada de elementos do grafo(i), que representa um conjunto estendido a ciclo obtido. O segundo passo é analisar cada conjunto estendido a ciclo para verificar a executabilidade da associação. Esta análise é feita aplicando-se a heurística proposta por Frankl e otimizando-a segundo o Capítulo 4, pelo algoritmo a seguir.

```
void det_conj_ciclo_exec(k, conj, arco_assoc, var_assoc, i, intera)
int k, conj;
PAIRINT * arco_assoc;
BITVECTOR * var_assoc;
int i;
{
    atualiza = 1;
    erro = 0;
    top_s = conj_est_ciclo[conj].top;
    n_exec = 0;
    do
    {
        if (atualiza)
        {
            ini_comp();
            atualiza = 0;
        }
    }
```

```

/* no ultimo com certeza ja encontrou arco e acabou = true */
get_elem_conj_ciclo(conj,&top_s,&elem,&prox_elem);
reset_all(&var_pred);
if (retr_pred(elem->infosuc.num_no_G,&pred_expr,&var_pred,pred))
{
    /* possui um predicado */
    mk_nl_address(&cecciclo.cam);
    push_address(elem,&cecciclo.cam);
    cecciclo.next = (CNJESTCICLO*)NULL;
    if ((elem->infosuc.num_no_G+1) < prox_elem->infosuc.num_no_G)
        /* predicado precisa valer false */
    {
        CNJESTCICLO *conjtrue = NULL;
        if (test_bit(elem->infosuc.num_no_G,&NLacos))
        {
            /* e' de um laço aplicar heurísticas */
            if (testa_contido_bitvector(&var_pred,var_assoc))
                conjtrue = get_cam_loop(conj,elem,var_assoc,&var_pred);
        }
        atualiza = (conjtrue== NULL);
        n_exec = get_exec_conj(conjtrue,&cecciclo,elem->infosuc.num_no_G,
                               &pred_expr,&var_pred,pred,conj);
    }
    else
        /* predicado precisa valer true, mesmo se for de um laço */
        n_exec = get_exec_conj(&cecciclo,NULL, elem->infosuc.num_no_G,
                               &pred_expr,&var_pred,pred,conj);
    if (n_exec)
    {
        /* e' não executável, então eliminar o conjunto estendido a ciclo */
        elimina_conj_ciclo(conj,elem->infosuc.num_no_G,k,i);
    }
}
else
{
    /* no' não possui predicado apenas executa os comandos do no' */
    erro = (exec_simb_no(elem->infosuc.num_no_G) == -1);
}
/* acaba quando o conjunto e' não executável, ou qdo o arco
da associação foi encontrado ou quando aconteceu erro na execução
simbólica */
}while (!((elem->infosuc.num_no_G == arco_assoc->abs &&
prox_elem->infosuc.num_no_G == arco_assoc->coor) || n_exec || erro));
}

```

get_elem_conj_ciclo(conj,top_s,elem,prox_elem): Obtém elemento (elem) e o próximo nó do caminho dado por conj_est_ciclo[conj]. Atualiza top_s para apontar para o próximo elemento do caminho.

reset_all(var_pred): torna todos os bits (variáveis) de var_pred igual a 0.

int retr_pred(no,pred_expr,var_pred,pred): recupera para o nó dado, o predicado correspondente, que contém: var_pred (variáveis do predicado), pred_expr (expressão do predicado na forma postfix, e pred (o predicado propriamente dito); retorna 0, se infono[i].num_pred = 0.

As variáveis ceciclo e conjtrue representam, respectivamente, um ciclo através do nó i, tal que este ciclo é igual a {(i)}; e um ponteiro para uma lista de ciclos através de um laço, tal que estes ciclos redefinem variáveis do predicado do laço e não redefinem as variáveis da associação.

As funções test_bit e esta_contido manipulam vetores de bits e verificam, respectivamente, se o bit correspondente a elem->infosuc.num_no_G em NLacos vale 1 e se o conjunto var_pred está contido no conjunto var_assoc.

CONJESTCICLO *get_cam_loop(conj,elem,var_assoc,var_pred): percorre o grafo(i) procurando por caminhos através do nó dado por elem que sejam livres de definição com respeito a var_assoc e que redefinam alguma das variáveis de var_pred. Novamente, o grafo(i) é utilizado pela facilidade dada pelo conjunto deff.

int get_exec_ciclo: esta função verifica a executabilidade do conjunto estendido a ciclo. Por exemplo, se não existirem caminhos a serem percorridos na hipótese do predicado ser avaliado false (ou true), para que o conjunto seja executável, o predicado precisa ser na verdade avaliado true (ou false). Esta função chama o módulo de avaliação de

predicados. Se o sistema estiver interagindo com o usuário as possibilidades acima serão enviadas para a tela.

elimina_conj_ciclo(conj,elem->infosuc_num_no_G,k,i): elimina o conjunto estendido a ciclo tornando bit conj igual a 1, num vetor CONJDESCON, indicando que conj_est_ciclo[conj] pode ser eliminado. Se todos os elementos de conj_est_ciclo puderem ser eliminados a associação, assim como os caminhos que a cobrem, serão eliminados. A eliminação consiste em colocar uma marca nos elementos não executáveis alterando os arquivos PDUPATHS.TES e PUASSOC.TES, e escrever a associação e caminhos correspondentes nos arquivos de caminhos e associações não executáveis.

int exec_simb_no(elem->infosuc_num_no_G): executa os comandos de um nó, retornando 1 se aconteceu algum erro. Nesse caso, a análise do conjunto terminará. Se o sistema estiver interagindo com o usuário, este poderá entrar valores para as variáveis do programa, inicializando o vetor COMP_SIMB, antes da execução dos comandos do nó.

4) O módulo det_cam_nexec

Esse módulo tem como função determinar caminhos não executáveis. Tem como entrada um ponteiro para o caminho a ser analisado. Desse caminho é obtido uma sequência representando a sequência de nós do GFC correspondente ao caminho dado. O módulo analisa cada nó da sequência ativando a execução simbólica dos seus comandos, e avaliando os predicados se infono[nó da sequência].num_pred > 0. Para executar o primeiro nó do caminho, os elementos de COMP_SIMB são inicializados com valores indeterminados. Se existir algum predicado não satisfeito, o caminho é um padrão e será eliminado, bem como todos os caminhos que incluirem o padrão determinado.

Se o caminho foi selecionado pelo usuário, então uma interação existe; os nós e predicados associados, bem como os resultados da avaliação são apresentados pelo sistema, e o usuário poderá avaliar os

predicados, inicializar COMP_SIMB antes da execução de cada nó, determinar não executabilidade, etc.

5.3 Exemplo de utilização das rotinas implementadas

Nesta seção apresentam-se um exemplo de utilização e uma avaliação das rotinas implementadas. O exemplo apresentado é o mesmo utilizado no Capítulo 4, Figura 4.1.

As telas e mensagens do sistema são diferenciadas das respostas do usuário que estão em negrito. Os comentários sobre a operação das rotinas estão em itálico.

Para início da sessão é necessário que as entradas mencionadas produzidas pelo módulo POKERNEL da POKE-TOOL e que os arquivos estejam no mesmo diretório em que se encontra a unidade em teste.

Primeiramente será feita a eliminação automática; as telas correspondentes e várias mensagens serão enviadas.

Apolo a geracao de casos de teste

Determinacao automatica de nao executabilidade

Eliminacao automatica de associacoes

grafo1

grafo2

grafo3

grafo8

Eliminacao automatica de caminhos

grafo1

grafo2

grafo3

grafo8

Como resultado dessa eliminação, foram alterados os arquivos *PDU PATHS.TES* e *PUASSOC.TES*, que contêm, respectivamente, os caminhos e as associações requeridas, e produzidos os arquivos *PDUNEXEC.TES* e *ASSOCNEXEC.TES*, conforme pode ser visto abaixo.

arquivo pdupaths.tes

CAMINHOS REQUERIDOS PELO CRITERIO TODOS POT-DU-CAMINHOS.

Caminhos requeridos pelo Grafo(1)

- * 1) 1 3 4 9
- * 2) 1 3 4 5 7 8 4
- 3) 1 3 4 5 6 8 4
- * 4) 1 2 4 9
- 5) 1 2 4 5 7 8 4
- * 6) 1 2 4 5 6 8 4

Caminhos requeridos pelo Grafo(2)

- 7) 2 4 9
- 8) 2 4 5 7 8 4
- 9) 2 4 5 6 8 4

Caminhos requeridos pelo Grafo(3)

- 10) 3 4 9
- 11) 3 4 5 7 8 4
- 12) 3 4 5 6 8 4

Caminhos requeridos pelo Grafo(8)

- 13) 8 4 9
- 14) 8 4 5 7 8
- 15) 8 4 5 6 8

ASSOCIACOES REQUERIDAS PELOS CRITERIOS TODOS POT-USOS E POT-USOS/DU.

Associacoes requeridas pelo Grafo(1)

- 1) <1,(1,3),{ concorre, concurso, i, finalist }>
- * 2) <1,(4,9),{ concorre, concurso, i, finalist }>
- 3) <1,(5,7),{ concorre, concurso, i, finalist }>
- 4) <1,(8,4),{ concorre, concurso, finalist }>
- 5) <1,(5,6),{ concorre, concurso, i, finalist }>
- 6) <1,(1,2),{ concorre, concurso, i, finalist }>

Associacoes requeridas pelo Grafo(2)

- 7) <2,(4,9),{ premiado }>
- 8) <2,(5,7),{ premiado }>
- 9) <2,(8,4),{ premiado }>
- 10) <2,(5,6),{ premiado }>

Associacoes requeridas pelo Grafo(3)

- 11) <3,(4,9),{ premiado }>
- 12) <3,(5,7),{ premiado }>
- 13) <3,(8,4),{ premiado }>
- 14) <3,(5,6),{ premiado }>

Associacoes requeridas pelo Grafo(8)

- 15) <8,(4,9),{ i }>
- 16) <8,(7,8),{ i }>
- 17) <8,(5,7),{ i }>
- 18) <8,(6,8),{ i }>
- 19) <8,(5,6),{ i }>

.arquivo pathsnexec.tes

POTENCIAIS DU-CAMINHOS NAO EXECUTAVEIS.

- 1) 1 3 4 9
- 4) 1 2 4 9
- 2) 1 3 4 5 7 8 4
- 6) 1 2 4 5 6 8 4

.arquivo assocnexec.tes

ASSOCIACOES NAO EXECUTAVEIS

- 2) <1,(4,9),{ concorre, concurso, i, finalist }>

Existem nesse programa ainda 4 caminhos e uma associação não executáveis que não foram identificados automaticamente pelo sistema.

O caminho 2 4 5 6 8 4 e a associação <2,(5,6),{premiado}> são não executáveis, pois após a execução do nó 2, i = 1; finalist = 10; premiado = 0; é impossível satisfazer i <= premiado para passar pelo nó 6, i é redefinida, mas está sendo sempre incrementada. Veja abaixo como a interação com o usuário vai ajudar a avaliar o predicho. Uma vez determinada como não executável a associação (2,(5,6),premiado), o caminho 2 4 5 6 8 4 é eliminado automaticamente.

.seleciona o grafot a ser analisado

Apoio a geracao de casos de teste

Determinacao iterativa de nao executabilidade

Nos que possuem definicao de variaveis e grafot

no 1

no 2

no 3

no 8

Seleciona numero do no i ou retorna(R:r)

==> 2

.seleciona opcao para o grafot - 2

- a. Visualiza/Seleciona associacao
- b. Visualiza/Seleciona pot-du-cam
- c. Mantem padroes
- d. Retorna para menu principal

Entre com a opcao desejada

==> a

Mensagens -----

.seleciona associacao

- 1. <2,(4,9),{premiado}>
- 2. <2,(5,7),{premiado}>
- 3. <2,(8,4),{premiado}>
- 4. <2,(5,6),{premiado}>

Seleciona numero da associacao ou retorna(R:r)

==> 4

.seleciona operacao para a associacao

Associacao selecionada: <2,(5,6),{premiado}>

- a. Visualiza/Seleciona conjuntos estendidos a ciclo
- b. Elimina associacao
- c. Executa caso de teste
- d. Avalia caso de teste
- e. Retorna

Entre com a opcao desejada:

==> a

Mensagens -----

.mostra conjuntos estendidos a ciclo

Conjuntos estendidos a ciclo para a associacao

<2,(5,6),{premiados}, para o grafo 2.

1. 2 4 5 6 8 4

Seleciona numero do conjunto ou retorna(R|r)

==> 1

. analisa o conjunto escolhido

Conjunto estendido a ciclo 2 4 5 6 8 4

Digite i para introducao de fatos associados ao no 2

Terminada a execucao dos comandos do no 2

Digite qualquer tecla para continuar

Pred(4) = (i<=finalist)

Pred(4) = false

L4 = { }

Pred(4) = true

L4 = { 4 5 6 8 4, 4 5 7 8 4, 4 }

Para a associacao ser executada e' necessario avaliar predicado true

Avalia predicado do no ou retorna(A!a//R!r)

==> a

Digite i para introducao de fatos associados ao no 4

Terminada a execucao de comandos do no 4

Digite qualquer tecla para continuar ...

mostra resultado da avaliacao do predicado

Variaveis do predicado

i = indeterminado

finalist = indeterminado

O predicado nao pode ser avaliado

Digite qualquer tecla para continuar ...

Digite i para introducao de fatos associados ao no 4

continua a análise

Conjunto estendido a ciclo 2 4 5 6 8 4

Pred(5) = (i<=premiado)

Pred(5) = false

L5 = { }

Pred(5) = true

L5 = { 5 }

Para a associação ser executada é necessário avaliar predicado false

Avalia predicado do no ou retorna(A:a//R:r)

==> a

Digite i para introdução de fatos associados ao no 5

Terminada a execução de comandos do no 5

Digite qualquer tecla para continuar

mostra resultado da avaliação do predicado

Variáveis do predicado

i = indeterminado

premiado = 0

O predicado não pode ser avaliado

Digite qualquer tecla para continuar ...

Digite i para introdução de fatos associados ao no 5

i

. o usuário avalia o predicado do nó 5, porque sabe que a variável i está sendo sempre incrementada. Note que durante a introdução de fatos, o sistema pede primeiro valores para as variáveis do predicado; se nenhum valor for fornecido o sistema pedirá então que o usuário avale o predicado.

Introdução de fatos

Iniciar valor das variáveis no nó 5

Digite o nome da variável, seu valor e o tipo segundo tabela abaixo

- a.real
- b.inteiro
- c.char
- d.booleano

Tecle <ret> para encerrar

var: tipo: valor:

Como avaliar predicado do nó 5

Pred(5) = (i<=premiado) = false

A associação é não executável

Os caminhos 2 4 9, 3 4 9 e 3 4 5 7 8 4, têm predicados que não são satisfeitos, mas não serão detectados, porque no início da execução dos comandos do nó 2, os valores de i e finalist estão indeterminados. Abaixo pode-se ver como a interação com o usuário resolve o problema.

.seleciona opção para o grafol - 3

- a. Visualiza/Seleciona associação
- b. Visualiza/Seleciona pot-du-cam
- c. Mantém padrões
- d. Retorna para menu principal

Entre com a opcao desejada

==> b

Mensagens -----

.seleciona o caminho

1. 3 4 9
2. 3 4 5 7 8 4
3. 3 4 5 8 4

Seleciona numero do potencial du-caminho ou retorna(R|r)

==> 2

. analisa predicados do caminho

Potencial du caminho: 3 4 5 7 8 4

Digite i para introducao de fatos associados ao no 3

Terminada a execucao dos comandos do no 3

Digite qualquer tecla para continuar ...

Pred(4) = (i<=finalist)

Para a caminho ser executavel predicho precisa valer true

Avalia predicho do no ou retorna(A:a//R:r)

==> a

Digite i para introducao de fatos associados ao no 4

Terminada a execucao de comandos do no 4

Digite qualquer tecla para continuar ...

.mostra resultado da avaliaçao do predicado

Variaveis do predicado

i = indeterminado

finalist = indeterminado

O predicado nao pode ser avaliado

Digite qualquer tecla para continuar ...

Digite i para introducao de fatos associados ao no 4

.continua a análise, esse predicado não influí no caminho

Conjunto estendido a ciclo 3 4 5 7 8 4

Pred(5) = (i<=premiado)

Para o caminho ser executavel predicado precisa valer false

Avalia predicado do no ou retorna(A!a//R!r)

==> a

Digite i para introducao de fatos associados ao no 5

Terminada a execucao de comandos do no 5

Digite qualquer tecla para continuar

.mostra resultado da avaliação do predicado

Variaveis do predicado

i = indeterminado

premiado = 5

O predicado nao pode ser avaliado

Digite qualquer tecla para continuar

Digite i para introducao de fatos associados ao no 5

i

. o usuário fornece o valor de i no nó 5.

Introducao de fatos

Iniciarizar valor das variaveis no no 5

Digite o nome da variavel, seu valor e o tipo segundo tabela abaixo

a.real

b.inteiro

c.char

d.booleano

Tecle <ret> para encerrar

var: i tipo: b valor:1

. mostra resultado da avaliação

Variaveis do predicado

i = 1

premiado = 5

O predicado foi avaliado true

Digite qualquer tecla para continuar...

O caminho e' nao executavel

. para o caminho 2 4 9 o usuário pode fornecer o valor de i no nó 2; por exemplo.

Potencial du-caminho: 2 4 9

Digite i para introducao de fatos associados ao no 2

i

Introducao de fatos

Iniciarizar valor das variaveis no no 2

Digite o nome da variavel, seu valor e o tipo segundo tabela abaixo

a.real

b.inteiro

c.char

d.booleano

Tecle <ret> para encerrar

var: i tipo: b valor: 1

Tecle <ret> para encerrar

var:finalist tipo: b valor: 10

Terminada a execucao de comandos do no 2

Digite qualquer tecla para continuar ...

Potencial du-caminho: 2 4 9

Pred(4) = (i<=finalist)

Para o caminho ser executavel predicado precisa valer false

Avalia predicado do no ou retorna(A!a//R:r)

==> a

Digite i para introducao de fatos associados ao no 4

Terminada a execucao de comandos do no 4

Digite qualquer tecla para continuar ...

mostra resultado da avaliação do predicado

Variaveis do predicado

i = 1

finalist = 10

O predicado foi avaliado true

Digite qualquer tecla para continuar...

O caminho e' nao executavel

Analogamente, determina-se que o caminho 3 4 9 é não executável.

5.4 Considerações Finais

Neste capítulo apresentaram-se os principais módulos que implementam as facilidades incorporadas à POKE-TOOL para tratamento de não executabilidade, e um exemplo de utilização desses módulos.

Pode-se notar que a fase interativa pode auxiliar o usuário a selecionar dados de teste para associações e caminhos executáveis, pois indica como os predicados devem ser satisfeitos.

O exemplo apresentado ilustra como é importante a fase de interação com o usuário; 50% dos caminhos e associações foram detectados em cada fase; 100% foram detectados pelo sistema. As Tabelas 5.1a e 5.1b apresentam os resultados obtidos para o exemplo e também os resultados (% de determinação de não executabilidade) para outras duas rotinas do "benchmark".

Para melhor avaliar o desempenho das rotinas implementadas, elas estão sendo testadas para as 29 rotinas que compõem o "benchmark". Os resultados de duas rotinas testadas --- APPEND e GETFNS --- são apresentados no Apêndice B. Para algumas das rotinas, a eficiência das heurísticas foi muito baixa. Um exemplo é a rotina COMMAND que não possui laços e a não executabilidade é dependente do contexto. Para outras, a limitação é dada pela implementação. Os predicados não podem ser avaliados porque possuem chamadas de procedimentos, manipulação de ponteiros e strings, etc.

Tabela 5.1: Avaliação das Rotinas Implementadas

a)

	caminhos n. exec.	automati- mente	interati- vamente	total
getfns	9	0(0%)	9 (100%)	9
append	22	20(90%)	2 (10%)	22
excap4	8	4(50%)	4 (50%)	8

b)

	assoc. n.exec.	automati- mente	interati- vamente	total
getfns	8	0(0%)	8 (100%)	8
append	11	11(100%)	0 (0%)	11
excap4	2	1(50%)	1 (50%)	2

Esses dados preliminares mostram que as facilidades propostas para determinar não executabilidade, heurísticas de Frankl e extensões, e padrões não executáveis, são muito eficientes. A limitação encontrada está na execução dos comandos e avaliação dos predicados. Isso ressalta a importância da introdução de fatos pelo usuário. Neste trabalho o objetivo é avaliar o desempenho de técnicas para determinar executabilidade; estudos estão sendo conduzidos para, numa próxima etapa, introduzir técnicas de avaliação simbólica mais poderosas para conseguir um melhor desempenho das facilidades propostas.

CAPÍTULO 6

CONCLUSÕES E TRABALHOS FUTUROS

6.1 Conclusões

A maioria dos programas possui caminhos não executáveis, até mesmo os bem formulados, como os programas do benchmark, utilizados para investigar e ilustrar os conceitos utilizados nesta dissertação.

A existência de caminhos não executáveis, além de aumentar o custo das atividades de teste, introduz muitos problemas para a condução dessas atividades; por exemplo, decidir se um determinado conjunto de casos de teste satisfaz ou não um determinado critério de teste; a relação de inclusão (ordem parcial) entre os critérios estruturais de teste é sensivelmente alterada, etc. Saliente-se ainda que, em geral, é indecifável se um dado caminho é ou não executável. A importância e necessidade de se incorporar facilidades de manipulação de caminhos não executáveis nas ferramentas que automatizem as atividades de teste é evidente.

Este trabalho identificou as linhas gerais de pesquisa nesta área. Na literatura existem basicamente três linhas de pesquisa, apresentadas no Capítulo 2: Woodward, Hedley e Hennel, que tratam de caracterizar como os caminhos não executáveis são gerados; outros, que visam prever e estimar a probabilidade de um caminho ser não executável, entre esses destacam-se os estudos de Malevris que visam determinar qual o relacionamento entre o número de predicados de um caminho e sua executabilidade; e o trabalho de Frankl, heurísticas para determinar não executabilidade.

Neste trabalho realizou-se um estudo sobre o problema de caminhos não executáveis nas atividades de teste. O estudo foi conduzido tendo por base os trabalhos citados e a experiência obtida durante a condução de um "benchmark", constituído de 29 rotinas implementadas em C, do livro "Software Tools in Pascal" de Kernighan e Plauger, utilizando-se a ferramenta de teste POKE-TOOL. Para esse "benchmark", determinaram-se manualmente os caminhos e associações não executáveis, exigidos pelos critérios Potenciais Usos, para cada uma das rotinas. O estudo realizado consistiu, à maneira dos trabalhos mencionados, em caracterizar, prever e determinar aspectos de não executabilidade e incorporar facilidades à ferramenta POKE-TOOL, para lidar com esses aspectos.

Na mesma linha dos trabalhos de Hedely e Hennel, agruparam-se em categorias as principais causas de não executabilidade encontradas nos caminhos das rotinas que compõem o "benchmark"; entre as principais causas de não executabilidade encontram-se os laços e flags que controlam esses laços.

Foram obtidos vários modelos de estimativa bastante razoáveis para estimar a variável resposta -número de caminhos não executáveis-, notando-se que várias características do programa, tais como: número de comandos de decisão, número de variáveis, número de nós com definição de variáveis, e número de potenciais-du-caminhos, influenciam no número de caminhos não executáveis do programa.

Também foram explorados vários modelos com o objetivo de validar as afirmações de Malevris. Pôde-se validar essas afirmações -- quanto maior o número de predicados do caminho menor a probabilidade dele ser executável -- para os potenciais-du-caminhos no benchmark considerado.

Algumas facilidades para tratamento de não executabilidade foram incorporadas à ferramenta POKE-TOOL, acrescentando-lhe algumas funções. As principais facilidades implementadas são: eliminação de caminhos e

associações não executáveis, determinação de associações não executáveis pela aplicação de heurísticas, determinação de caminhos não executáveis através de inconsistências, utilização de padrões não executáveis, etc. Foram propostos o uso de extensões a ciclo e duas otimizações para a implementação da heurística de Frankl.

Caracterizar os caminhos não executáveis foi muito importante para poder, dado um programa, dizer qual a probabilidade dele conter muitos caminhos não executáveis, para obter programas com um número reduzido de caminhos não executáveis, e para determinar heurísticas e facilidades para tratamento de não executabilidade.

A utilização de padrões de não executabilidade, permite que sejam eliminados muitos caminhos, cuja não executabilidade é originada pela mesma causa. Automatizar a eliminação de padrões é muito importante pois o número de caminhos que contêm o padrão pode ser muito grande.

Avaliando-se o desempenho das rotinas implementadas para algumas das rotinas do benchmark, a heurística de Frankl e as facilidades incorporadas à POKE-TOOL mostraram-se eficientes, determinando-se não executabilidade para muitos casos. As limitações encontradas para alcançar um melhor desempenho foram consequência da execução simbólica e da avaliação de predicados que apresentam inúmeras dificuldades de implementação. Mas como foi apresentado no Capítulo 5, essas limitações podem ser bastante reduzidas, com a participação do usuário que pode avaliar predicados e introduzir fatos. Além disso, a fase interativa pode auxiliar o usuário a selecionar dados de teste para associações e caminhos executáveis, pois indica como os predicados devem ser satisfeitos ao longo dos caminhos analisados.

6.2 Trabalhos Futuros

Muitos dos resultados obtidos são resultantes de estudos preliminares e constituem-se em motivação para a realização de estudos mais rigorosos e de novos trabalhos. Existem várias direções de pesquisa futura para não executabilidade de caminhos sob os três pontos de vista abordados nesta tese: caracterização, previsão e determinação de não executabilidade. Alguns trabalhos de prosseguimento deste são listados a seguir:

- 1) Sob o ponto de vista de caracterização de caminhos não executáveis, estudar mais profundamente as principais causas de não executabilidade, e baseados nestas causas, propor técnicas para escrever programas que gerem um número menor de caminhos não executáveis e propor novas heurísticas para determinar não executabilidade.
- 2) Realizar uma análise estatística mais rigorosa e completa, para se obter melhores modelos para estimar o número de caminhos não executáveis de um programa, dada as várias características do programa, e para estimar a probabilidade de um caminho ser executável, dado o seu número de predicados.
- 3) Continuar a avaliação e validação das rotinas incorporadas à POKE-TOOL e das facilidades implementadas, para todas as rotinas que compõem o benchmark. Além disso, estudar e implementar técnicas de avaliação mais poderosas para melhorar o desempenho das rotinas implementadas, e continuar os estudos para implementar facilidades à POKE-TOOL para geração de dados de teste, depuração e manutenção.
- 4) Realizar os estudos conduzidos nesse trabalho e os trabalhos propostos acima, para um outro benchmark. Para isso, a ferramenta POKE-TOOL já está sendo configurada para também permitir sua utilização em outras linguagens de programação tais como PASCAL, FORTRAN, COBOL.

REFERÊNCIAS

- [CLA76] L. Clarke, "A System to Generate Test Data and Symbolically Execute Programs", IEEE Trans. Soft. Eng., Vol. SE-2, September 1976.
- [CLA86] L. Clarke, A. Podgurski, D. J. Richardson e S. J. Zeil, "A Comparison of Data Flow Path Selection Criteria", in Proc. of the 8th Int'l Conf. on Software Engineering, pp 244-251, Aug. 1985.
- [CHA91a] M. L. Chaim, J. C. Maldonado, M. Jino, "Projeto/Implementação de Uma Ferramenta de Teste de Software," Relatório Técnico DCA/FEE/UNICAMP- Campinas,SP, Brasil, 1991.
- [CHA91b] M. L. Chaim, J. C. Maldonado, M. Jino, Manual de Configuração da POKE-TOOL, Relatório Técnico DCA/FEE/UNICAMP- Campinas,SP, Brasil, 1991.
- [CHA91c] M. L. Chaim, J. C. Maldonado, M. Jino, Manual do Usuário da POKE-TOOL Relatório Técnico DCA/FEE/UNICAMP- Campinas,SP, Brasil, 1991.
- [CHA91d] 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.
- [CHU87] T. Chusho, "Test Data Selection and Quality Estimation Based on Concept of Essential Branches for Paths Testing", IEEE Trans. Soft. Eng. Vol. SE-13, No.5, May 1987, pp 509-517.
- [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.
- [FRA88] F. G. Frankl, E. J. Weyuker, "An Applicable Family of Data Flow Testing Criteria," IEEE, Trans. on Software Eng. Vol.14, No.10, pp 1483-1498, Oct. 1988.
- [GHE87] C. Ghezzi e M. Jazayeri, Programming Languages Concepts, John Wiley and Sons, segunda edição, New York, 1987.
- [HEC77] Hecht, M. S., "Flow Analysis of Computer Programs", North Holland, New York, 1977.
- [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.
- [HER76] Herman, P.M., "Flow Analysis Approach to Program Testing", The Australian Computer Journal, Vol.8, No.3, Nov.1976, pp.92-96.
- [HOW77] Howden W. E., "Symbolic testing and Dissect Symbolic Evaluation System", IEEE Trans. Soft. Eng., Vol. SE-3, No.4, pp 266-278, July 1977.
- [HOW87] Howden W. E., Functional Program Testing and Analysis, McGrawHill, USA, 1987.
- [KER81] B. W. Kernighan e P. J. Plauger, Software Tools in Pascal. Addison-Wesley Publishing Company, Reading, Massachusetts, 1981.
- [LAS83] Laski, J. W. and Korel, B., "A Data Flow Oriented Program Testing Strategy", IEEE Trans. Software Eng., Vol. SE-9, No.3, May 1983 pp 347-354.

- [McC76] T. McCabe, "A Software Complexity Measure," IEEE Trans. Software Eng., Vol. 2, pp. 308-320, Dez. 1976.
- [MAY90] Mayhauser, A., Software Engineering - Methods and Management, Academic Press Inc, USA, 1990.
- [MAL88a] J. C. Maldonado, M.L Chaim, 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.
- [MAL88b] J. C. Maldonado, M. L. Chaim, M. Jino, "Resultados do Estudo de uma Família de Critérios de Teste de Programas Baseada em Fluxo de Dados," Relatório Interno - DCA/FEE/UNICAMP - RT/DCA-001/88 - Campinas,SP,Brasil.
- [MAL89a] J. C. Maldonado, M. L. Chaim, M. Jino, "Arquitetura de uma Ferramenta de Teste de Apoio aos Critérios Potenciais Usos, in Proc. XXII Congresso Nacional de Informática São Paulo, SP, Brasil, Set. 1989.
- [MAL89b] J. C. Maldonado, M. L. Chaim, M. Jino, "Feasible Potential Use Criteria Analysis," Technical Report- DCA/FEE/UNICAMP - RT/DCA-001/89 - Campinas, SP, Brasil, 1989.
- [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, M. Jino, "Potential Uses Criteria Complexity Analysis," Technical Notes - DCA/FEE/UNICAMP - TN/DCA-001/91 - Campinas, SP, Brasil 1991.

- [MAL91c] J.C. Maldonado, M.L. Chaim, S. R. Vergílio, M. Jino, "Critérios Potenciais Usos: Uma Contribuição 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 and A.Veavers, "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.
- [NTA84] Ntafos, S. C., "On Required Element Testing", Trans. Software Eng., Vol.SE-10, Nov. 1984, pp 795-803.
- [NTA88] Ntafos, S.C., "A Comparison of Some Structural Testing Strategies", IEEE- Trans. Sof. Eng., Vol.4, No.6, Jun. 1988, pp 868-873.
- [PRE87] R. B. Pressman, Software Engineering: a practitioner's approach, Second Edition, New York, MacGraw-Hill, 1987.
- [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.
- [RYA76] T.A. Ryan Jr., B.L. Joiner and B.F. Ryan, MINITAB - Student Handbook, Wadsworth Publishing Company, California, 1976.

- [RAM76] C. V. Ramamoorthy, F. Ho Siu-Bun, W. T. Chen, "On Automated Generation of Program Test Data", IEEE Trans. Soft. Eng. Vol.SE-2, No. 4, December 1976.
- [URA88] Ural, U. and Yang, B., "A Structural Test Selection Criterion", Information Processing Letters, Vol. 28, Jul. 1988., pp 157-163.
- [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.
- [WYE86] E.J. Weyuker, "Axiomatizing Software Test Data Adequacy", IEEE Trans. Soft. Eng., Vol.SE -12, No.2, Aug 1984.
- [WEY88a] E.J. Weyuker, "The Evaluation of Program-Based Software Test Data Adequacy Criteria", IEEE Trans. Soft. Eng., Vol.SE -16, No.2, Feb.1988.
- [WEY88b] E.J. Weyuker, "An Empirical Study of the Complexity of Data Flow Testing", in Proc.of Second Workshop on Software, Verification and Analysis (TAV2), Banff, Canada, Jul. 1988.
- [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.
- [WOO80] M.R.Woodward, D.Hedley, M.A. Hennel, "Experience with Path Analysis and Testing of Programs", IEEE Trans. Soft. Eng., Vol SE-6, May 1980, pp 278-286.
- [YAT89] D.F. Yates, N. Malevris, "Reducing the Effects of Infeasible Paths in Branch Testing", ACM Computer Surveys, pp 48-54, 1989.

APÊNDICE A

PRINCIPAIS MODELOS OBTIDOS NA ANÁLISE DE ASPECTOS DE EXECUTABILIDADE DE CAMINHOS NAS ROTINAS DO "BENCHMARK" APLICADO

Este apêndice apresenta uma visão geral da análise e resultados do ponto de vista de executabilidade de caminhos obtidos da aplicação de um benchmark para avaliação empírica dos critérios Potenciais Usos, utilizando-se a ferramenta de teste POKE-TOOL. Primeiramente são apresentados detalhadamente os dados coletados para cada uma das rotinas. As outras seções apresentam os melhores modelos obtidos para avaliar a influência de várias características do programa na executabilidade dos caminhos e para validar a hipótese de Malevris, da influência do número de predicados de um caminho na sua executabilidade, para as rotinas do benchmark aplicado.

A.1 Potenciais-du-Caminhos X Número de Predicados

Primeiramente, apresentam-se para cada uma das rotinas do "benchmark" as tabelas contendo o número de potenciais du-caminhos executáveis e não executáveis e respectivos números de predicados a eles associados. Deve-se notar as rotinas ressaltadas no Capítulo 3: ARCHIVE, GETCMD, DODASH, MAKEPAT, SUBST, TRANSLIT, EDIT e GETONE; as três primeiras com nenhum caminho não executável, as restantes com um número muito grande de caminhos não executáveis.

amatch

	0	1	2	3	4	5	6	7	8	9	10	11	12	tot
nex	0	0	1	7	7	4	5	9	10	4	8	6	0	61
exc	0	1	6	2	4	3	2	2	3	3	0	0	0	26
tot	0	1	7	9	11	7	7	11	13	7	8	6	0	87

append

	0	1	2	3	4	5	6	7	8	9	10	tot
nex	0	0	1	5	2	3	9	2	0	0	0	22
exc	1	1	3	2	1	2	3	3	3	2	0	21
tot	1	1	4	7	3	5	12	5	3	2	0	43

archive

	0	1	2	3	4	5	6	7	8	9	10	11	tot
nex	0	0	0	0	0	0	0	0	0	0	0	0	0
exc	4	0	2	0	2	1	1	2	2	1	2	0	17
tot	4	0	2	0	2	1	1	2	2	1	2	0	17

change

	0	1	2	3	4	5	6	7	tot
nex	0	1	0	1	0	0	0	0	2
exc	0	7	5	5	4	3	1	0	25
tot	0	8	5	6	4	3	1	0	27

ckg lob

	0	1	2	3	4	5	6	7	8	9	tot
nex	0	1	0	0	0	2	1	2	1	0	7
exc	0	11	2	3	3	0	1	0	0	0	20
tot	0	12	2	3	3	2	2	2	1	0	27

command

	0	1	2	3	tot
nex	0	2	17	0	19
exc	13	14	15	0	42
tot	13	16	32	0	61

cmp

	0	1	2	3	4	5	6	7	tot
nex	0	0	0	1	1	1	0	0	3
exc	0	0	1	1	2	1	4	0	9
tot	0	0	1	2	3	2	4	0	12

compare

	0	1	2	3	4	5	6	7	8	9	10	11	12	tot
nex	0	0	0	0	1	2	4	8	1	10	0	4	0	30
exc	0	0	1	0	0	2	5	6	5	2	2	0	0	23
tot	0	0	1	0	1	4	9	14	6	12	2	4	0	53

compress

	0	1	2	3	4	5	6	7	tot
nex	0	2	0	1	2	5	0	0	10
exc	0	2	0	4	7	8	3	0	24
tot	0	4	0	5	9	13	3	0	34

dodash

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	tot
nex	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exc	0	2	1	2	2	3	1	2	1	0	4	2	1	0	21
tot	0	2	1	2	2	3	1	2	1	0	4	2	1	0	21

edit

	0	1	2	3	4	5	6	7	8	9	10	11	tot
nex	0	1	1	2	11	27	57	64	51	33	14	0	261
exc	0	2	5	4	8	13	27	17	11	7	2	0	96
tot	0	3	6	6	19	40	84	81	62	40	16	0	357

entab

	0	1	2	3	4	5	6	7	8	tot
nex	0	2	2	6	3	11	7	2	0	33
exc	0	1	2	10	9	11	2	2	0	37
tot	0	3	4	16	12	22	9	4	0	70

expand

	0	1	2	3	4	5	6	tot
nex	0	0	2	4	0	9	0	15
exc	0	3	2	5	0	0	0	10
tot	0	3	4	9	0	9	0	25

getcmd

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	tot
nex	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exc	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	0	119
tot	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	0	119

getdef

	0	1	2	3	4	5	6	7	8	9	10	11	tot
nex	0	2	2	2	1	1	1	0	0	0	0	0	9
exc	0	4	5	6	5	9	2	4	2	1	2	0	40
tot	0	6	7	8	6	10	3	4	2	1	2	0	49

gtext

	0	1	2	3	4	5	6	7	8	9	tot
nex	0	0	0	0	1	6	2	0	1	0	10
exc	0	1	1	3	3	1	1	1	1	0	12
tot	0	1	1	3	4	7	3	1	2	0	22

getfn

	0	1	2	3	4	5	6	tot
nex	0	0	1	1	2	3	0	7
exc	0	0	3	1	4	5	0	13
tot	0	0	4	2	6	8	0	20

getfns

	0	1	2	3	4	5	6	7	8	tot
nex	0	0	4	0	2	0	2	1	0	9
exc	0	10	7	3	4	1	0	0	0	25
tot	0	10	11	3	6	1	2	1	0	34

getlist

	0	1	2	3	4	5	6	7	8	tot
nex	0	0	2	2	22	3	19	16	0	64
exc	0	2	2	6	12	5	3	8	0	38
tot	0	2	4	8	34	8	22	24	0	102

getnum

	0	1	2	3	4	5	6	7	8	tot
nex	0	0	1	1	1	0	1	1	0	5
exc	0	0	1	1	1	0	1	3	0	7
tot	0	0	2	2	2	0	2	4	0	12

getone

	0	1	2	3	4	5	6	7	8	9	10	11	12	tot
nex	0	0	0	0	1	3	34	55	3	6	8	32	0	142
exc	0	0	0	0	1	1	19	3	1	2	8	0	0	35
tot	0	0	0	0	2	4	53	58	4	8	16	32	0	177

makepat

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	tot
nex	0	0	0	0	0	4	24	35	11	14	22	5	13	5	2	25	5	0	4	0	169
exc	0	2	0	0	0	2	12	13	5	6	12	4	9	7	0	9	3	0	0	0	84
tot	0	2	0	0	0	6	36	48	16	20	34	9	22	12	2	34	8	0	4	0	253

omatch

	0	1	2	3	4	5	6	tot
nex	0	7	1	0	11	2	0	21
exc	0	7	1	1	11	2	0	22
tot	0	14	2	1	22	4	0	43

optpat

	0	1	2	3	4	5	6	tot
nex	0	0	2	3	3	5	0	13
exc	1	0	2	1	1	3	0	8
tot	1	0	4	4	4	8	0	21

rquick

	0	1	2	3	4	5	6	7	8	9	10	tot
nex	0	0	0	0	0	5	0	10	16	2	0	33
exc	0	1	2	0	3	4	3	3	2	0	0	18
tot	0	1	2	0	3	9	3	13	18	2	0	51

spread

	0	1	2	3	4	5	6	7	8	tot
nex	0	0	3	1	7	2	0	1	0	14
exc	0	5	5	8	5	7	0	3	0	33
tot	0	5	8	9	12	9	0	4	0	47

subst

0	1	2	3	4	5	6	7	8	9	10	11	12	13	tot	
nex	0	0	0	0	4	12	21	30	20	24	65	47	14	0	237
exc	0	0	7	5	7	4	9	14	5	8	15	9	2	0	85
tot	0	0	7	5	11	16	30	44	25	32	80	56	16	0	322

translit

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	tot	
nex	0	0	0	0	1	2	5	7	17	22	21	23	30	26	22	15	9	5	2	0	207
exc	0	7	4	7	7	10	7	6	6	8	8	13	9	6	8	8	5	5	2	0	126
tot	0	7	4	7	8	12	12	13	23	30	29	36	39	32	30	23	14	10	4	0	333

unrotate

0	1	2	3	4	5	6	7	8	tot	
nex	0	1	1	7	7	6	3	4	0	29
exc	0	8	13	10	4	1	5	0	0	41
tot	0	9	14	17	11	7	8	4	0	70

A.2 Melhores Modelos para Avaliar a Influência de Várias Características do Programa no Número de Caminhos Não Executáveis

Foram determinados modelos para estimar a influência de várias características do programa, tais como número de comandos de decisão, número de variáveis, número de nós com definição de variáveis, número de potenciais-du-caminhos, etc, no número de casos de teste exigidos para os critérios Potenciais Usos básicos e no número de caminhos não executáveis do programa. A análise foi realizada inicialmente, para dois conjuntos de rotinas, um incluindo-se todas as 29 rotinas do benchmark e outro excluindo-se as rotinas recursivas AMATCH e RQUICK.

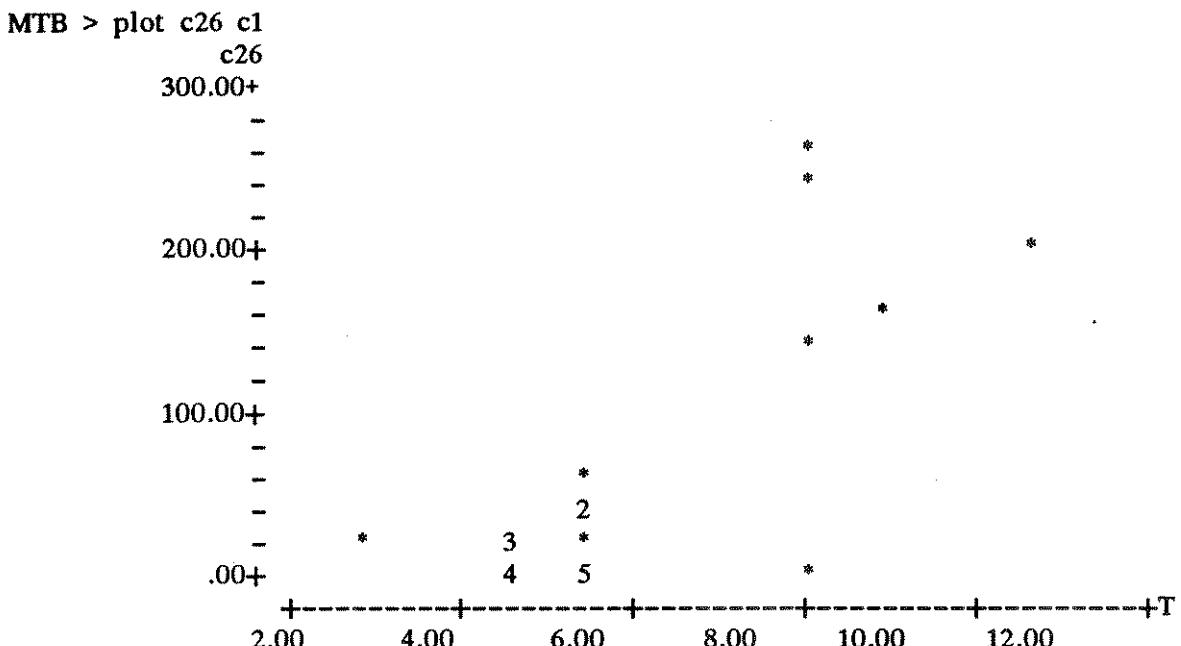
Os principais modelos obtidos para estimar a variável resposta número de casos de teste, encontram-se em [MAL91a]. Nessa seção encontram-se de forma mais detalhada algumas das características dos principais modelos obtidos para o critério todos-potenciais-du-caminhos para estimar a variável resposta número de caminhos não executáveis.

A.2.1 Variável de Controle: número de comandos de decisão (t)

```

MTB > omit 2 in c23, coor,c1-c4,c11,c12,c26,put c23,c1-c4,c11,c12,c26
MTB > omit 87 in c12, corr,c1-c4,c11,c26,c23,put c12,c1-c4,c11,c26,c23
MTB > choose 012 in c1,coor c2-c4,c11,c12,c26,c23,put c1-c4,c11,c12,c26,c23
MTB > omit 20 in c11,corr c1-c4,c12,c26,c23,put,c11,c1-c4,c12,c26,c23
MTB > print c1 c2 c3 c4 c11 c12 c26

```



```
MTB > regress c26 1 c1,c42,c42
```

THE REGRESSION EQUATION IS

$$C26 = -152 + 31.6t$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-151.78	35.43	-4.28
t	31.633	5.180	6.11

$$S = 51.21$$

R-SQUARED = 64.0 PERCENT

R-SQUARED = 62.3 PERCENT, ADJUSTED FOR D.F.

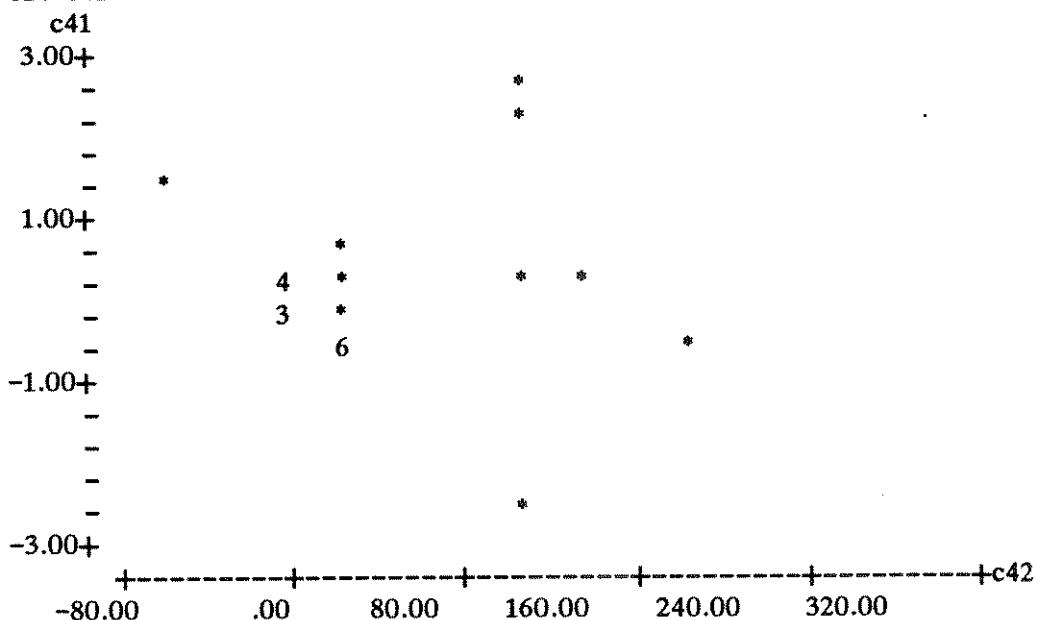
ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS-SS/DF
REGRESSION	1	97800	97800
RESIDUAL	21	55066	2622
TOTAL	22	152866	

DURBIN-WATSON STATISTIC = 1.69

A.2.2 Variável de Controle: número de variáveis (variav)

MTB > plot c14 c42



MTB > regress c26 2 c1 c2,c41,c42

THE REGRESSION EQUATION IS

$$C26 = -173 + 26.7 t + 8.37 \text{ variav}$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-173.28	32.67	-5.30
t	26.723	5.002	5.34
variav	8.373	3.293	2.54

$$S = 45.61$$

R-SQUARED = 72.8 PERCENT

R-SQUARED = 70.1 PERCENT, ADJUSTED FOR D.F.

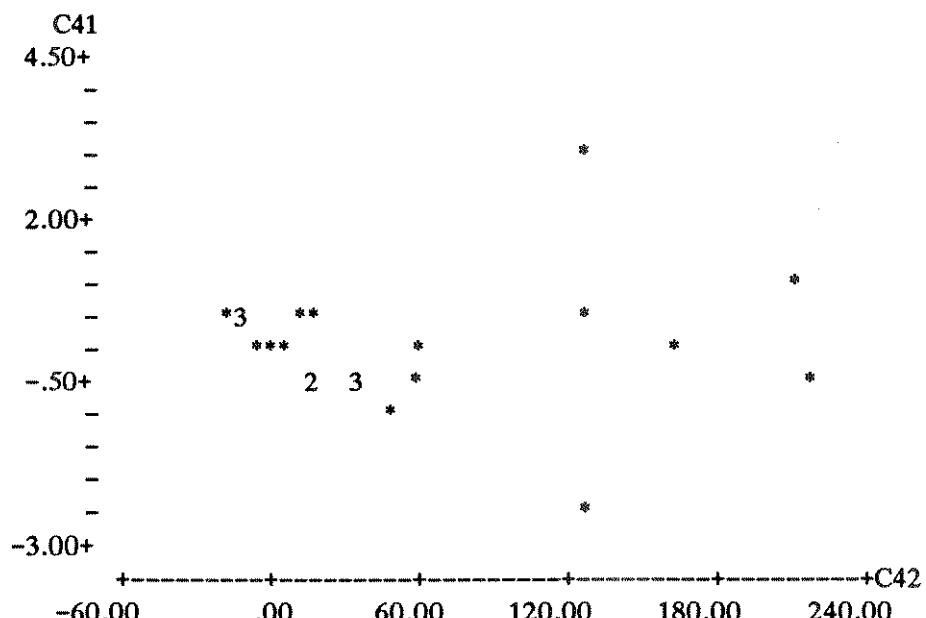
ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	111252	55626
RESIDUAL	20	41614	2081
TOTAL	22	152866	

DURBIN-WATSON STATISTIC = 1.99

A.2.3 Variável de Controle: número de definições (def)

MTB > plot c41 c42



MTB > regress c26 2 c1 c3,c41,c42

THE REGRESSION EQUATION IS

$$C26 = -154 + 23.6 t + 4.33 \text{ def}$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-154.17	30.55	-5.05
t	23.621	5.263	4.49
def	4.334	1.507	2.88

$$S = 44.14$$

R-SQUARED = 74.5 PERCENT

R-SQUARED = 72.0 PERCENT, ADJUSTED FOR D.F.

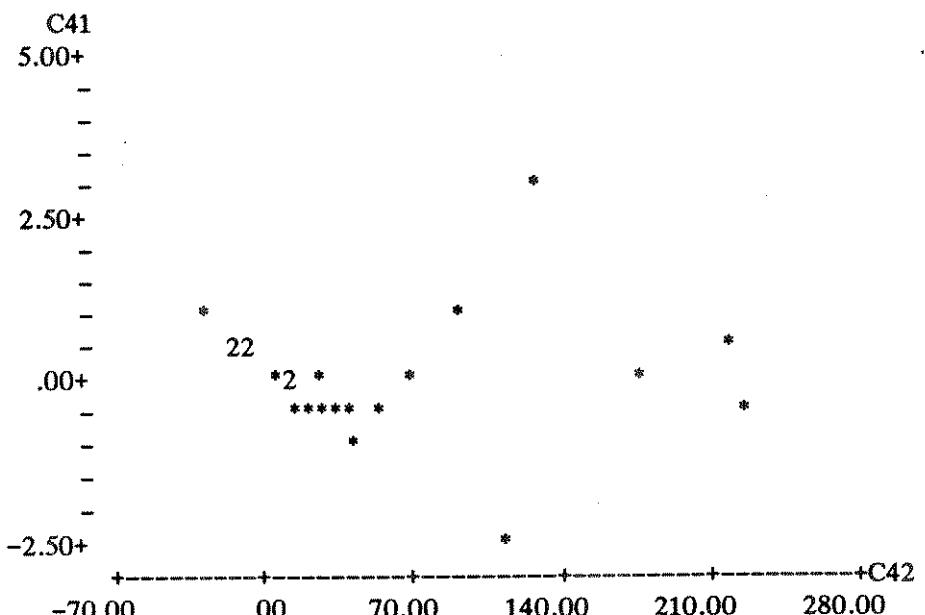
ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	113903	56952
RESIDUAL	20	38963	1948
TOTAL	22	152866	

DURBIN-WATSON STATISTIC = 2.26

A.2.4 Variável de Controle: número de nós com definição (nodef)

MTB > plot c41 c42



MTB > regress c26 2 c1 c4,c41,c42

THE REGRESSION EQUATION IS

$$C26 = -153 + 23.9 t + 7.66 \text{ nodef}$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-153.33	32.34	-4.74
t	23.909	5.811	4.11
nodef	7.655	3.351	2.28

$$S = 46.73$$

R-SQUARED = 71.4 PERCENT

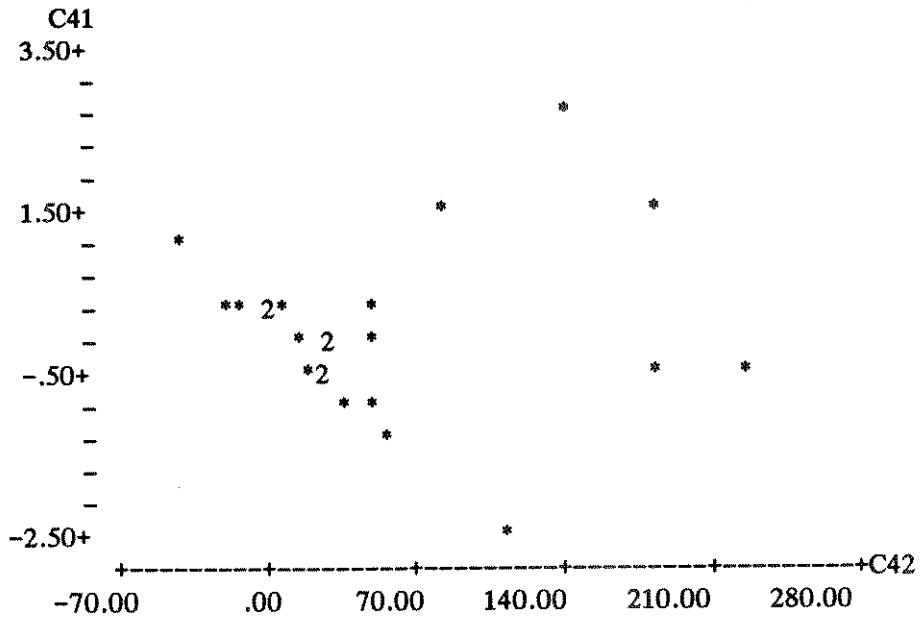
R-SQUARED = 68.6 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	109193	54597
RESIDUAL	20	43673	2184
TOTAL	22	152866	

DURBIN-WATSON STATISTIC = 2.11

MTB > plot c41 c42



MTB > omit 2 in c23, corr. c1-c4,c11,c12,c26,put c23,c1-c4,c11,c12,c26

MTB > omit 87 in c12, corr. c1-c4,c11,c26,c23,put c12,c1-c4,c11,c26,c23

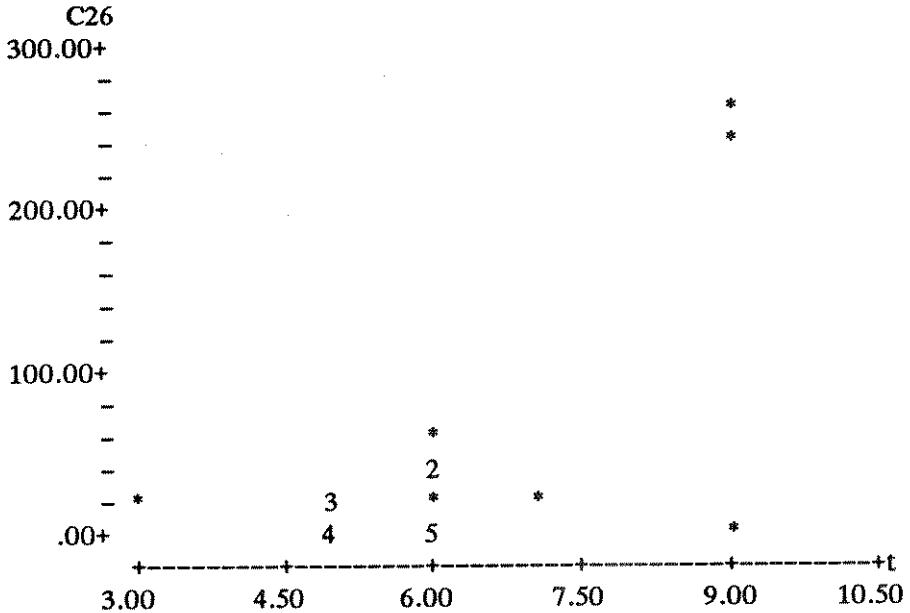
MTB > choose 0 11 in c1,corr c2-c4,c11,c12,c26,c23,put c1-c4,c11,c12,c26,c23

MTB > omit 39 in c23,corr c1-c4,c11,c12,c26,put c23,c1-c4,c11,c12,c26

MTB > omit 177 in c12, corr. c1-c4,c11,c26,c23,put c12,c1-c4,c11,c26,c23

MTB > print c1 c2 c3 c4 c11 c12 c26

MTB > plot c26 c1



**A.2.5 Variável de Controle: número de potenciais-du-caminhos
(ducam)**

MTB > regress c26 1 c12,c41,c42

THE REGRESSION EQUATION IS
C26 = - 14.3 + 0.766 ducam

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-14.321	1.679	-8.53
ducam	0.76629	0.01485	51.61

S = 6.226

R-SQUARED = 99.3 PERCENT

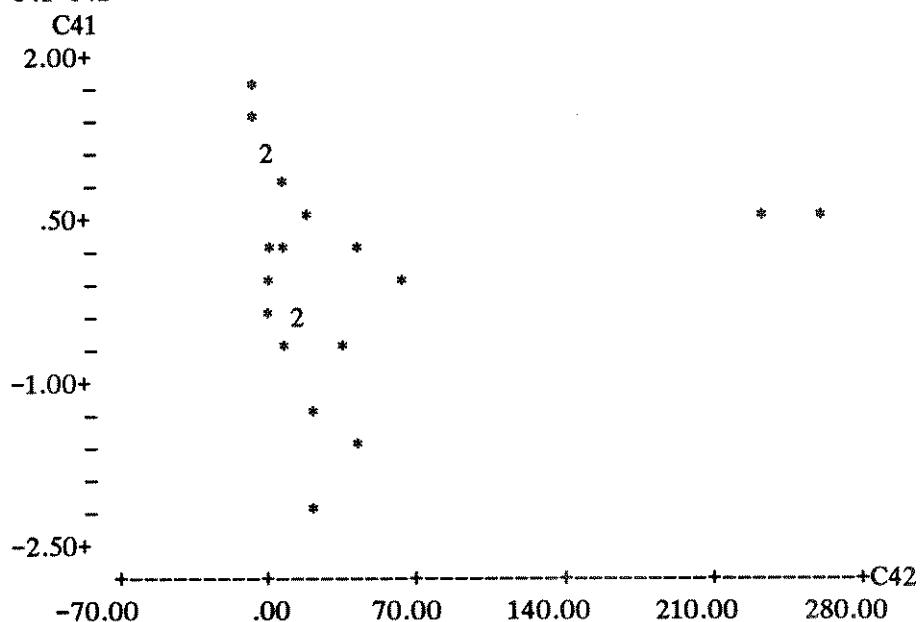
R-SQUARED = 99.3 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	1	103246	103246
RESIDUAL	19	737	39
TOTAL	20	103983	

DURBIN-WATSON STATISTIC = 2.87

MTB > plot c41 c42



A.3 MELHORES MODELOS OBTIDOS PARA AVALIAR A INFLUÊNCIA DO NÚMERO DE PREDICADOS DE UM CAMINHO NA SUA EXECUTABILIDADE

Nesta seção estão os principais modelos obtidos para validar a hipótese de Malevris, da influência do número de predicados de um caminho na sua executabilidade.

Novamente, para realizar a análise, foram considerados dois conjuntos de rotinas distintas, TOTAL com as 29 rotinas do benchmark e TOTAL-R, excluindo-se as duas rotinas recursivas (AMATCH e RQUICK). Foram considerados vários intervalos de valores de q , número de predicados do caminho $1 \leq q \leq 18$, $1 \leq q \leq 9$. Foram obtidos modelos polinomiais e exponenciais. A hipótese de Malevris pôde ser validada no intervalo de $1 \leq q \leq 9$.

Como apenas 5 rotinas SUBST, TRANSLIT, MAKEPAT, DODASH e GETCMD, contribuem com caminhos para $q \geq 12$, e para que a análise não ficasse influenciada pelo comportamento de uma rotina em especial também foram obtidos modelos excluindo-se as rotinas com $q \geq 12$; esses modelos assemelham-se aos modelos obtidos anteriormente, como descrito no Capítulo 3.

Primeiramente, apresenta-se uma síntese dos melhores modelos obtidos para o conjunto TOTAL, considerando os intervalos citados e, posteriormente, uma síntese dos melhores modelos para o conjunto TOTAL-R, que exclui as unidades recursivas.

A.3.1 Melhores Modelos para o Conjunto TOTAL

A.3.1.1 Considerando-se o intervalo $1 \leq q \leq 18$

1) Modelo Polinomial

MTB > regress c5 2 c1 c21, c10,c11

THE REGRESSION EQUATION IS
PropFEAS = 0.925 - 0.106 #q + 0.00419 C21

COLUMN	COEFFICIENT	ST. DEV.	T-RATIO =
		OF COEF.	COEF/S.D.
	0.92528	0.04892	18.91
#q	-0.10636	0.01207	-8.81
C21	0.0041942	0.0006341	6.61

S = 0.06114

R-SQUARED = 90.9 PERCENT

R-SQUARED = 89.6 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	0.52012	0.26006
RESIDUAL	14	0.05233	0.00374
TOTAL	16	0.57245	

FURTHER ANALYSIS OF VARIANCE

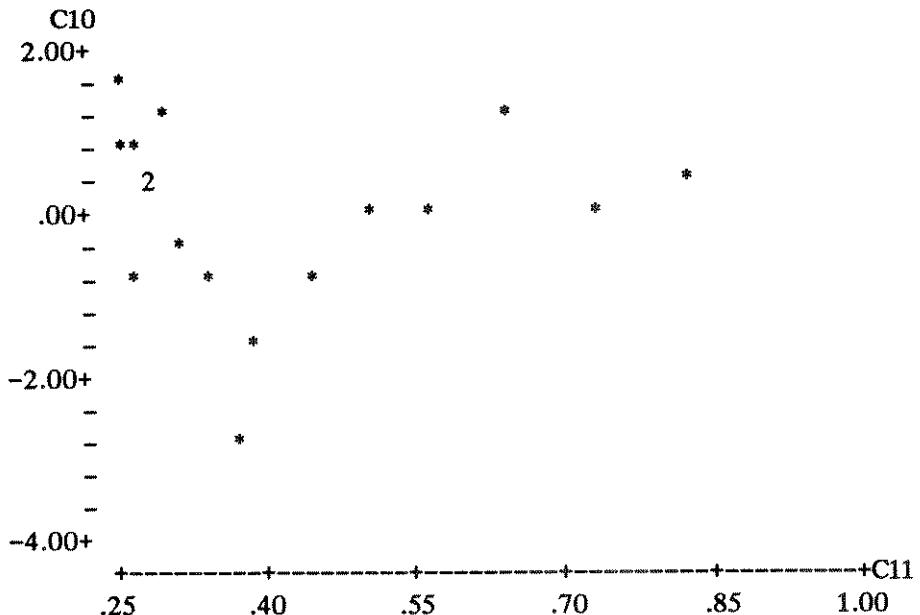
SS EXPLAINED BY EACH VARIABLE WHEN ENTERED IN THE ORDER GIVEN

DUE TO	DF	SS
REGRESSION	2	0.52012
#q	1	0.35658
C21	1	0.16355

DURBIN-WATSON STATISTIC = 1.32

MTB > plot c10 c11

1



2) Modelo Exponencial

MTB >

MTB > choose 0 16 c1, corr c2-c6,c21, put c1-c6,c21

MTB > regress c6 2 c1 c21, c10,c11

THE REGRESSION EQUATION IS

LePFEAS = 0.200 - 0.262 #q + 0.0116 C21

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	0.1996	0.1041	1.92
#q	-0.26199	0.02818	-9.30
C21	0.011552	0.001611	7.17

S = 0.1218

R-SQUARED = 92.0 PERCENT

R-SQUARED = 90.8 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	2.2254	1.1127
RESIDUAL	13	0.1928	0.0148
TOTAL	15	2.4183	

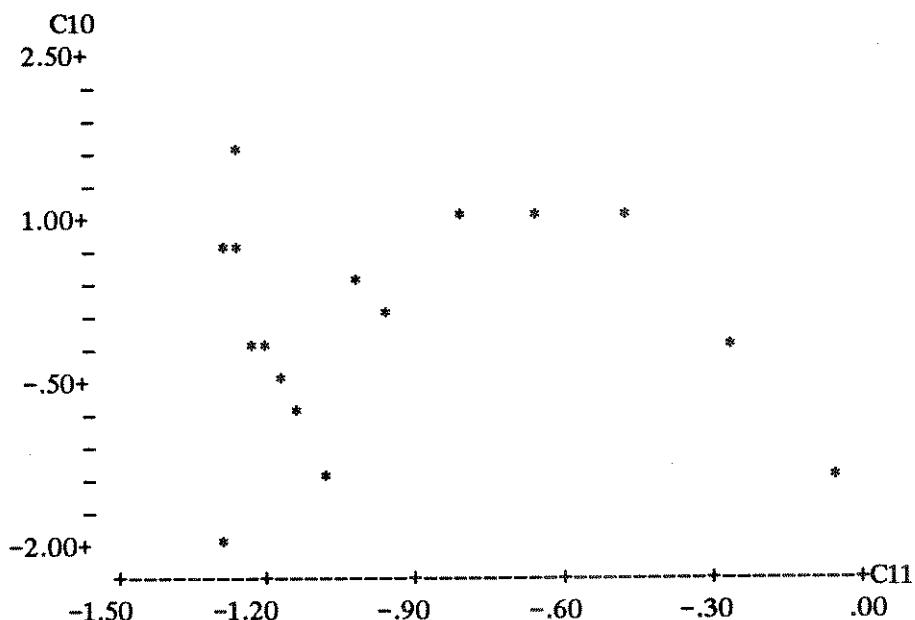
FURTHER ANALYSIS OF VARIANCE

SS EXPLAINED BY EACH VARIABLE WHEN ENTERED IN THE ORDER GIVEN

DUE TO	DF	SS
REGRESSION	2	2.2254
#q	1	1.4632
C21	1	0.7623

DURBIN-WATSON STATISTIC = 1.85

MTB > plot c10 c11



A.3.1.2 Considerando-se o intervalo $1 \leq q \leq 9$

1) Modelo Exponencial

MTB >

MTB > plot c6 c1

LePFEAS

.00+

-

*

-

*

-

*

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

-

.00

2.00

4.00

6.00

8.00

10.00

+-----+
#q

MTB >

MTB > regress c6 1 c1, c10,c11

THE REGRESSION EQUATION IS

LePFEAS = 0.0148 - 0.153 #q

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	0.01480	0.06905	0.21
#q	-0.15291	0.01227	-12.46

S = 0.09505

R-SQUARED = 95.7 PERCENT

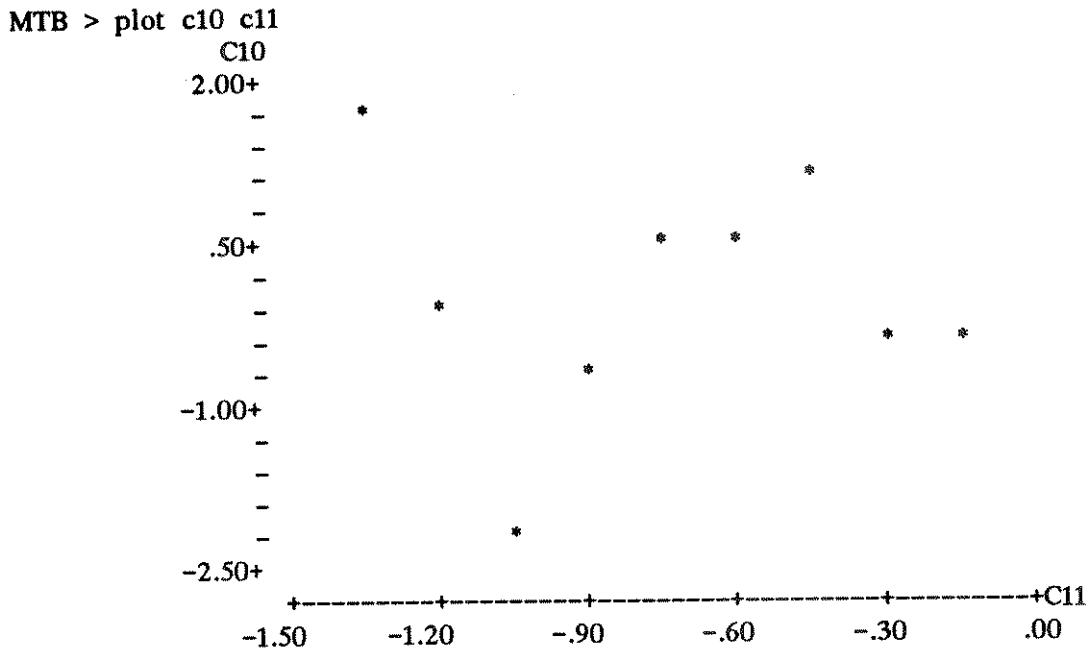
R-SQUARED = 95.1 PERCENT, ADJUSTED FOR D.F.

1

ANALYSIS OF VARIANCE

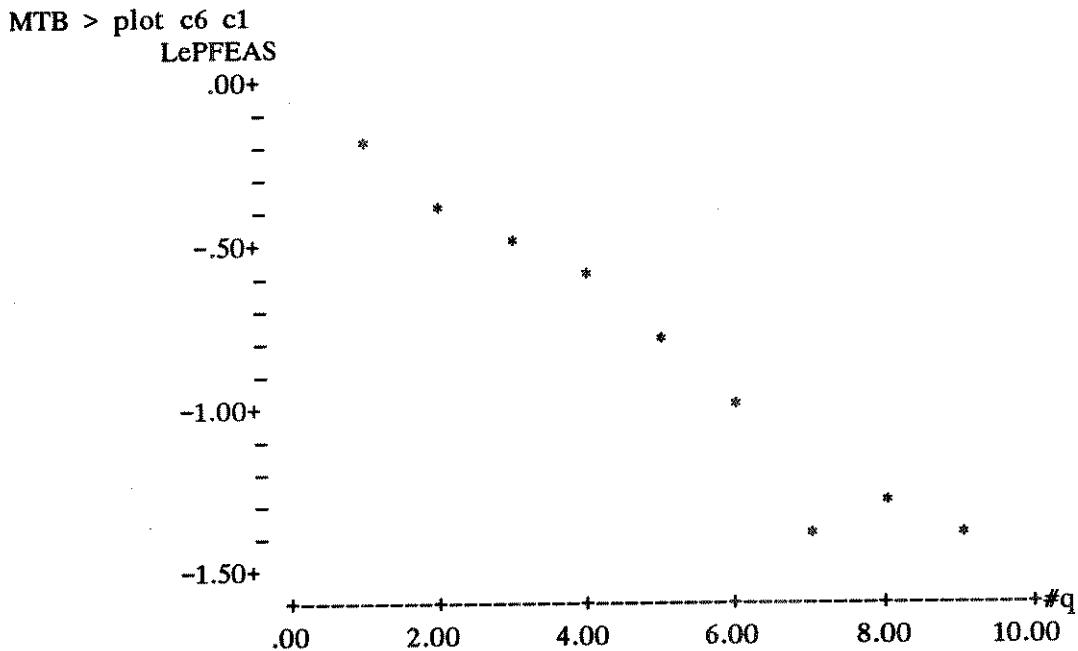
DUE TO	DF	SS	MS=SS/DF
REGRESSION	1	1.4029	1.4029
RESIDUAL	7	0.0632	0.0090
TOTAL	8	1.4662	

DURBIN-WATSON STATISTIC = 1.35



A.3.1.3. Considerando-se o intervalo $1 \leq q \leq 12$ e excluindo-se as 5 rotinas com $q \geq 12$.

1) Modelo Exponencial



MTB > regress c6 1 c1, c10,c11

THE REGRESSION EQUATION IS
LePFEAS = - 0.0302 - 0.164 #q

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-0.03021	0.07434	-0.41
#q	-0.16362	0.01321	-12.39

S = 0.1023

R-SQUARED = 95.6 PERCENT

R-SQUARED = 95.0 PERCENT, ADJUSTED FOR D.F.

1

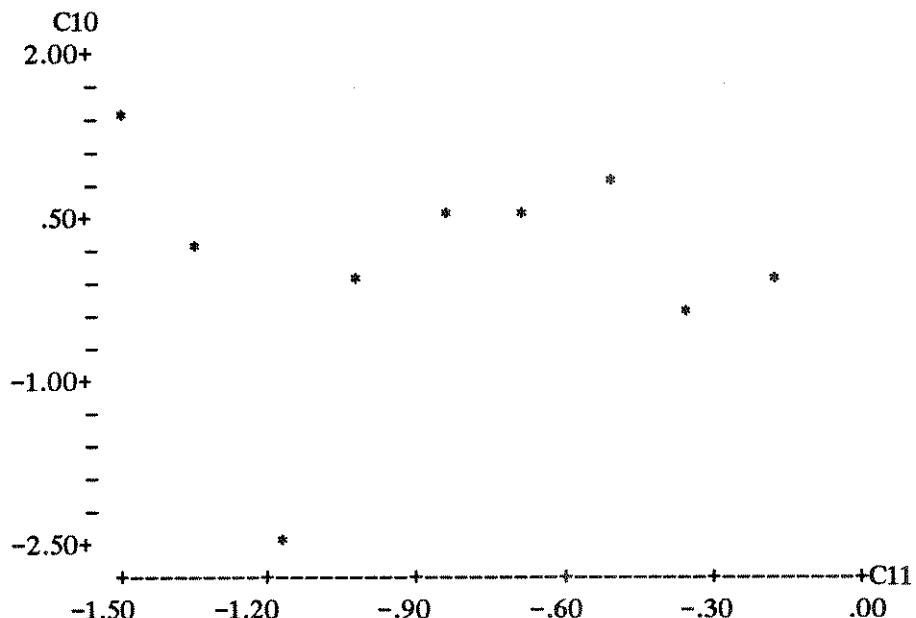
ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	1	1.6063	1.6063
RESIDUAL	7	0.0733	0.0105
TOTAL	8	1.6796	

R DENOTES AN OBS. WITH A LARGE ST. RES.

DURBIN-WATSON STATISTIC = 1.81

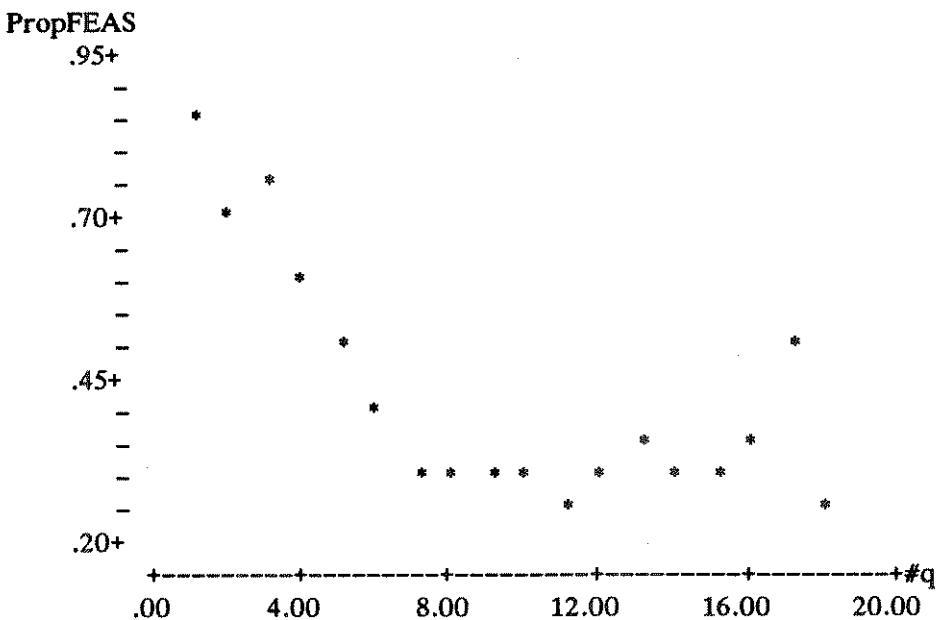
MTB > plot c10 c11



A.3.2 Melhores Modelos para o Conjunto TOTAL-R

A.3.2.1 Considerando-se o intervalo $1 \leq q \leq 18$

1) Modelo Polinomial



MTB > regress c5 2 c1 c21, c10,c11

THE REGRESSION EQUATION IS

$$\text{PropFEAS} = 0.926 - 0.104 \#q + 0.00407 \text{ C21}$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	0.92566	0.04969	18.63
#q	-0.10449	0.01226	-8.52
C21	0.0040749	0.0006441	6.33

$$S = 0.06210$$

R-SQUARED = 90.6 PERCENT

R-SQUARED = 89.2 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	0.51923	0.25962
RESIDUAL	14	0.05399	0.00386
TOTAL	16	0.57322	

FURTHER ANALYSIS OF VARIANCE

SS EXPLAINED BY EACH VARIABLE WHEN ENTERED IN THE ORDER GIVEN

DUE TO	DF	SS
REGRESSION	2	0.51923
#q	1	0.36485
C21	1	0.15438

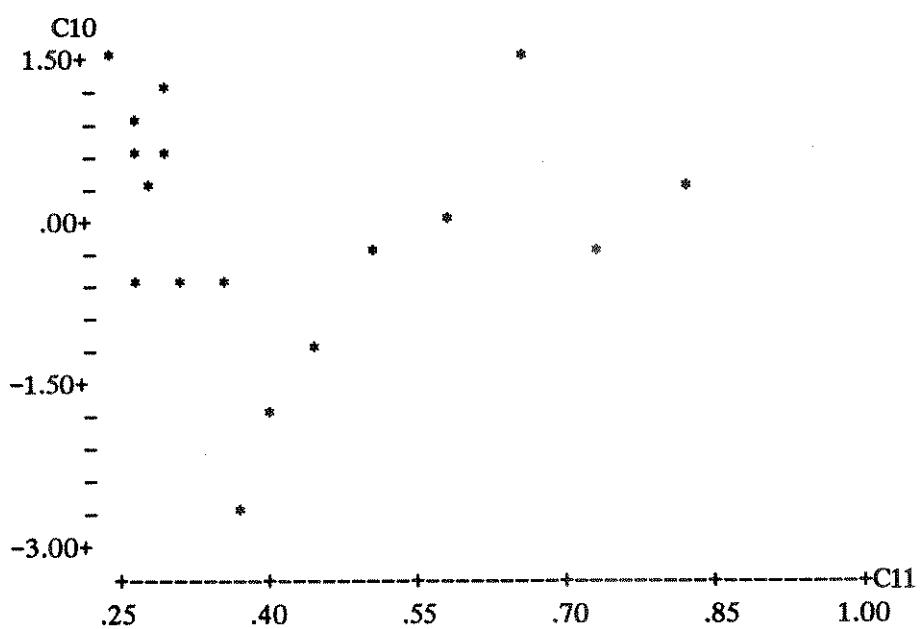
R DENOTES AN OBS. WITH A LARGE ST. RES.

X DENOTES AN OBS. WHOSE X VALUE GIVES IT LARGE INFLUENCE.

DURBIN-WATSON STATISTIC = 1.48

MTB > plot c10 c11

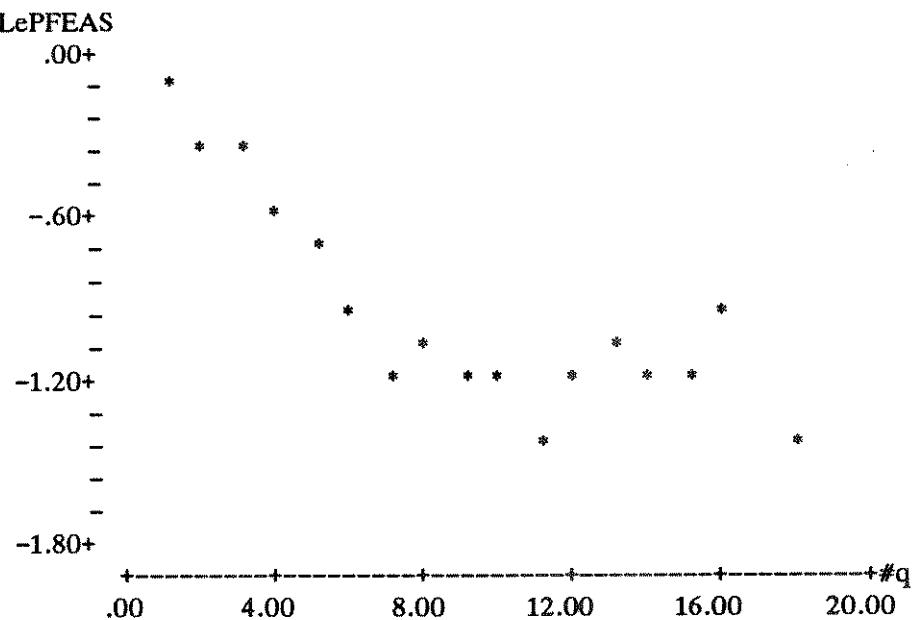
1



2) Modelo Exponencial

MTB > plot c6 c1

1



MTB >

MTB > choose 0 16 c1, corr c2-c6,c21, put c1-c6,c21

MTB > regress c6 2 c1 c21, c10,c11

THE REGRESSION EQUATION IS

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	0.1853	0.1029	1.80
#q	-0.25227	0.02787	-9.05
C21	0.010965	0.001594	6.88
S =	0.1205		

R-SQUARED = 92.0 PERCENT

R-SQUARED = 90.7 PERCENT, ADJUSTED FOR D.F.

ANALYSIS OF VARIANCE

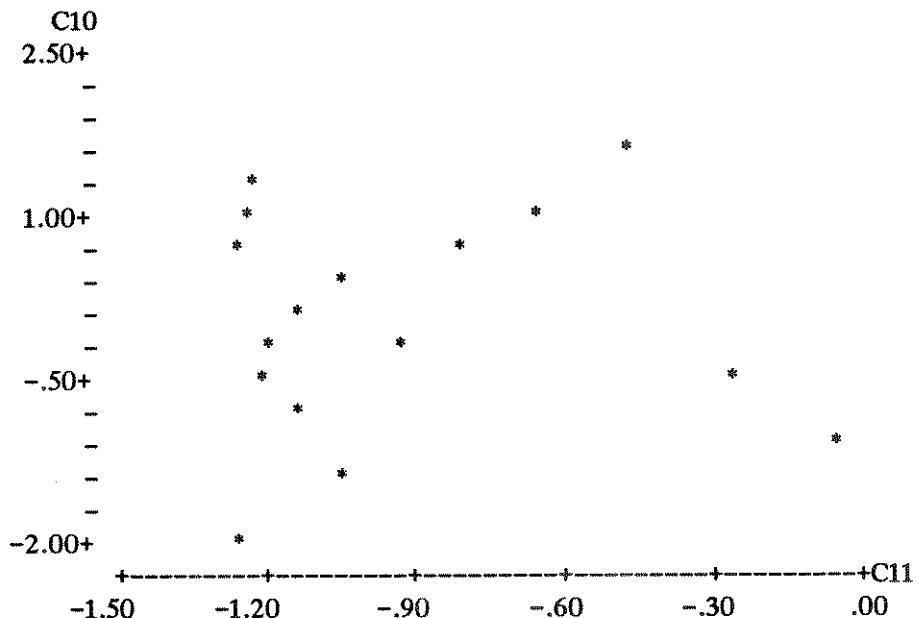
DUE TO	DF	SS	MS=SS/DF
REGRESSION	2	2.1617	1.0809
RESIDUAL	13	0.1886	0.0145
TOTAL	15	2.3504	

FURTHER ANALYSIS OF VARIANCE

SS EXPLAINED BY EACH VARIABLE WHEN ENTERED IN THE ORDER GIVEN

DUE TO	DF	SS
REGRESSION	2	2.1617
#q	1	1.4750
C21	1	0.6868

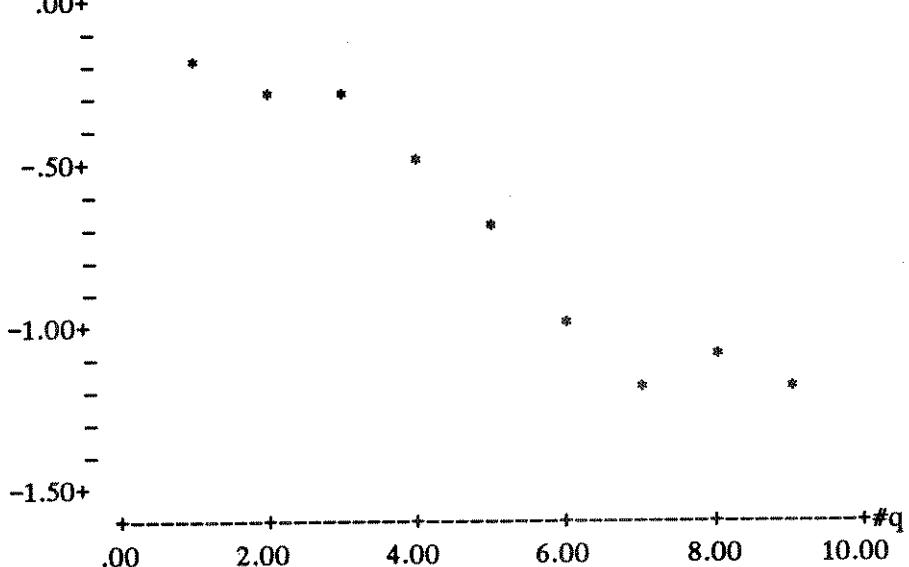
MTB > plot c10 c11



A.3.2.2 Considerando-se o intervalo $1 \leq q \leq 9$.

1) Modelo Exponencial

MTB > plot c6 c1
LePFEAS



MTB > regress c6 1 c1, c10,c11

THE REGRESSION EQUATION IS

$$\text{LePFEAS} = 0.0118 - 0.149 \#q$$

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	0.01185	0.07280	0.16
#q	-0.14936	0.01294	-11.55

$$S = 0.1002$$

R-SQUARED = 95.0 PERCENT

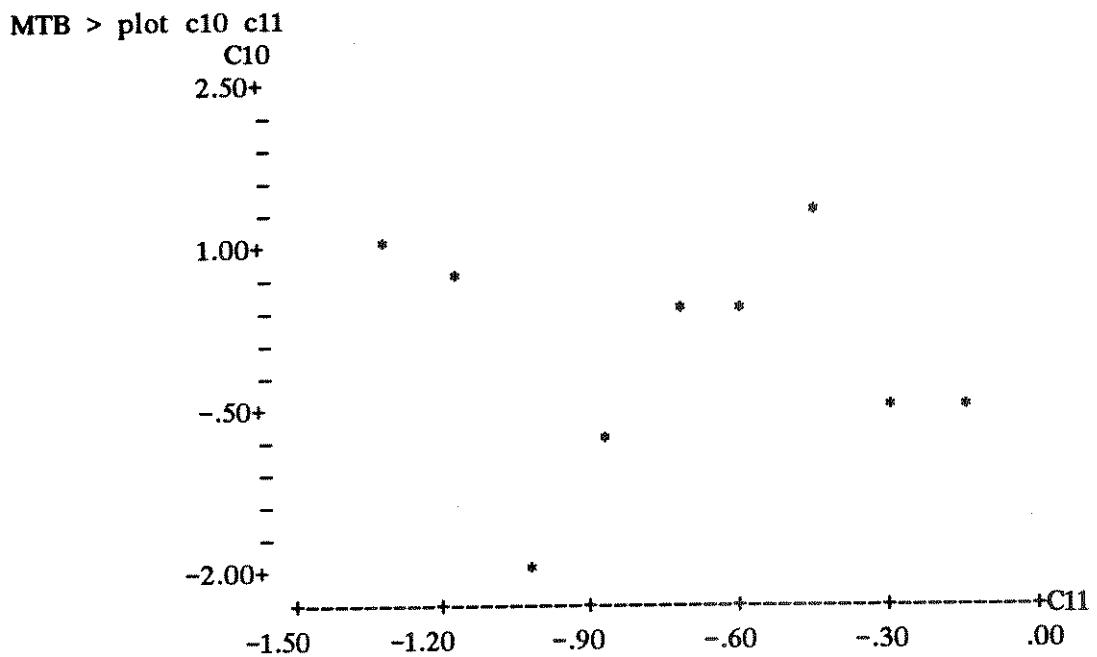
R-SQUARED = 94.3 PERCENT, ADJUSTED FOR D.F.

1

ANALYSIS OF VARIANCE

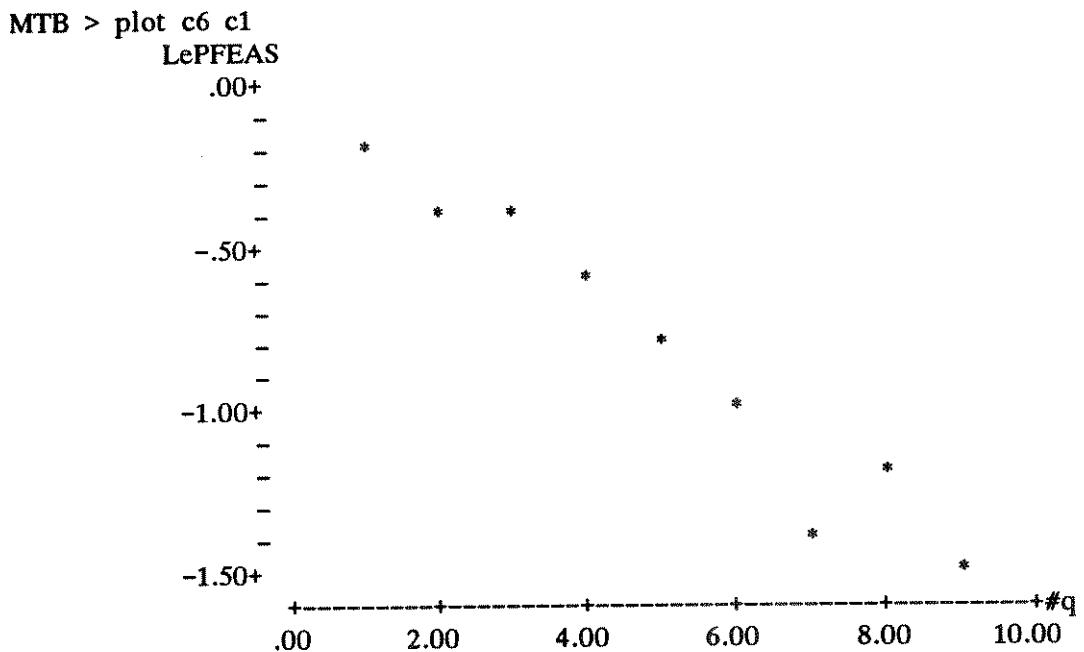
DUE TO	DF	SS	MS=SS/DF
REGRESSION	1	1.3385	1.3385
RESIDUAL	7	0.0703	0.0100
TOTAL	8	1.4088	

DURBIN-WATSON STATISTIC = 1.59



A.3.2.3 Considerando-se o intervalo $1 \leq q < 12$ excluindo-se as 5 rotinas com $q \geq 12$.

1) Modelo Exponencial



MTB > regress c6 1 c1, c10,c11

THE REGRESSION EQUATION IS
LePFEAS = - 0.0351 - 0.160 #q

COLUMN	COEFFICIENT	ST. DEV. OF COEF.	T-RATIO = COEF/S.D.
	-0.03508	0.08275	-0.42
#q	-0.16004	0.01471	-10.88

S = 0.1139

R-SQUARED = 94.4 PERCENT

R-SQUARED = 93.6 PERCENT, ADJUSTED FOR D.F.

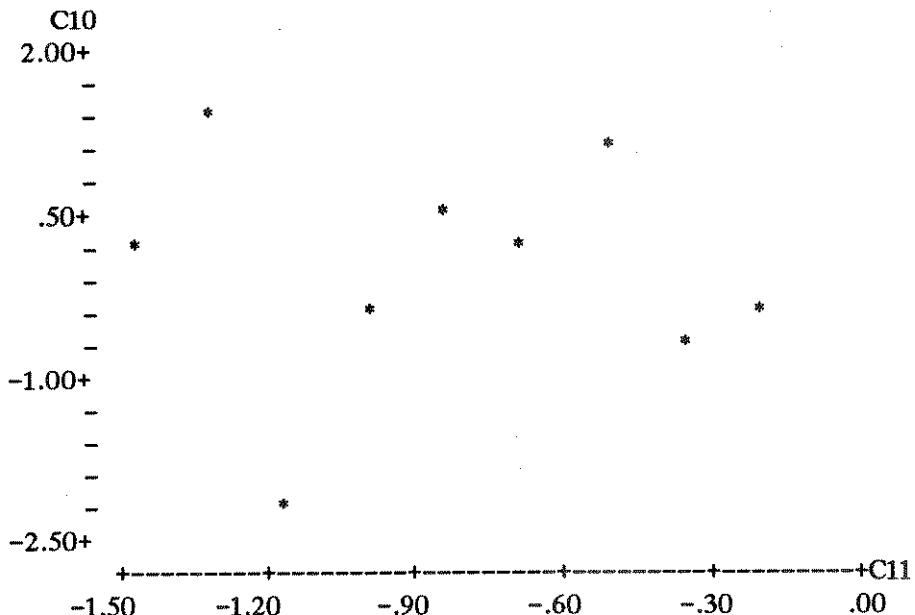
1

ANALYSIS OF VARIANCE

DUE TO	DF	SS	MS=SS/DF
REGRESSION	1	1.5367	1.5367
RESIDUAL	7	0.0908	0.0130
TOTAL	8	1.6276	

DURBIN-WATSON STATISTIC = 2.43

MTB > plot c10 c11



APÊNDICE B

EXEMPLOS DE UTILIZAÇÃO DAS FACILIDADES IMPLEMENTADAS

Este apêndice contém dois outros exemplos de utilização das rotinas incorporadas à POKE-TOOL. Para o exemplo do Capítulo 5, 50% dos elementos não executáveis foram determinados automaticamente e outros 50%, interativamente com o usuário. Os exemplos --- APPEND e GETFNS --- apresentados fazem parte do conjunto de rotinas que compõem o "benchmark"; as facilidades implementadas foram eficientes para as duas rotinas. Para a rotina APPEND, 100% das associações e 90% dos caminhos não executáveis foram determinados automaticamente. Para a rotina GETFNS, ao contrário, 100% dos caminhos e associações foram determinadas utilizando-se as facilidades implementadas, mas com o auxílio do usuário. Isso ressalta a limitação existente na execução simbólica e na avaliação de predicados e a importância da fase interativa implementada.

Como utilizado no Capítulo 5, as telas e mensagens do sistema são diferenciadas das respostas do usuário que estão em negrito. Os comentários sobre a operação das rotinas estão em itálico.

B.1 Rotina APPEND

Abaixo é apresentada uma versão instrumentada pela POKE_TOOL da rotina append. Os números da coluna da esquerda, representam o número do nó do grafo de fluxo de controle.

```
#define ponta_de_prova(num) if(printed_nodes % 10) (++printed_nodes; fprintf(p  
else {++printed_nodes; sprintf(path,"%d",num);}  
  
extern int curln;  
stcode append (line,glob)  
int line, glob;  
/* 1 */ {  
    FILE * path = fopen("path.tes","w");  
    static int printed_nodes = 0;  
    char inline[50];  
    stcode stat;  
    int done;  
    ponta_de_prova(1);  
    /* 1 */  
    /* 2 */ {  
        if(glob)  
        {  
            ponta_de_prova(2);  
            stat = 1;  
        }  
        else  
        {  
            ponta_de_prova(3);  
            curln = line;  
            stat = 2;  
            done = 0;  
            while(!done) && (stat == 2)  
            {  
                ponta_de_prova(4);  
                ponta_de_prova(5);  
                if(!getline(inline,stdin,50))  
                {  
                    ponta_de_prova(6);  
                    stat = 0;  
                }  
                else  
                {  
                    ponta_de_prova(7);  
                    if((inline[0] == '.') && (inline[1] == 13))  
                    {  
                        ponta_de_prova(8);  
                        done = 1;  
                    }  
                    else  
                    {  
                        ponta_de_prova(9);  
                    }  
                }  
            }  
        }  
    }  
}
```

```

/* 9 */           if(puttxt(inline) == 1)
/* 10 */           {
/* 10 */               ponta_de_prova(10);
/* 10 */               stat = 1;
/* 11 */           }
/* 11 */           ponta_de_prova(11);
/* 12 */       }
/* 12 */       ponta_de_prova(12);
/* 13 */   }
/* 13 */   ponta_de_prova(4);
/* 13 */   ponta_de_prova(14);
/* 14 */ }
/* 14 */ ponta_de_prova(15);
/* 14 */ ponta_de_prova(16);
fclose(path);
/* 15 */
/* 16 */ return (stat);
}

```

Primeiramente será feita a eliminação automática; as telas correspondentes e várias mensagens serão enviadas.

Apoio a geracao de casos de teste

Determinacao automatica de nao executabilidade

Eliminacao automatica de associacoes

- grafo1
- grafo2
- grafo3
- grafo5
- grafo6
- grafo8
- grafo9
- grafo10

Eliminacao automatica de caminhos

grafo1
grafo2
grafo3
grafo5
grafo6
grafo8
grafo9
grafo10

Como resultado dessa eliminação, foram alterados os arquivos PDUPATHS.TES e PUASSOC.TES, que contêm, respectivamente, os caminhos e as associações requeridas, e produzidos os arquivos PDUNEXEC.TES e ASSOCNEXEC.TES, conforme pode ser visto abaixo.

arquivo pdupaths.tes

>>> PDUPATHS.TES

CAMINHOS REQUERIDOS PELO CRITERIO TODOS POT-DU-CAMINHOS.

Caminhos requeridos pelo Grafo(1)

- * 1) 1 3 4 14 15 16
- 2) 1 3 4 5 7 9 11 12 13 4
- 3) 1 3 4 5 7 9 10 11 12 13 4
- 4) 1 3 4 5 7 8 12 13 4
- 5) 1 3 4 5 6 13 4
- 6) 1 2 15 16

Caminhos requeridos pelo Grafo(2)

- 7) 2 15 16

Caminhos requeridos pelo Grafo(3)

- * 8) 3 4 14 15 16
- 9) 3 4 5 7 9 11 12 13 4
- 10) 3 4 5 7 9 10 11 12 13 4
- 11) 3 4 5 7 8 12 13 4
- 12) 3 4 5 6 13 4

Caminhos requeridos pelo Grafo(5)

- 13) 5 7 9 11 12 13 4 14 15 16
- 14) 5 7 9 11 12 13 4 5
- 15) 5 7 9 10 11 12 13 4 14 15 16
- *16) 5 7 9 10 11 12 13 4 5
- 17) 5 7 8 12 13 4 14 15 16
- *18) 5 7 8 12 13 4 5
- 19) 5 6 13 4 14 15 16
- *20) 5 6 13 4 5

Caminhos requeridos pelo Grafo(6)

- 21) 6 13 4 14 15 16
- *22) 6 13 4 5 7 9 11 12 13
- *23) 6 13 4 5 7 9 10
- *24) 6 13 4 5 7 8 12 13
- *25) 6 13 4 5 6

Caminhos requeridos pelo Grafo(8)

- 26) 8 12 13 4 14 15 16
- *27) 8 12 13 4 5 7 9 11 12
- *28) 8 12 13 4 5 7 9 10 11 12
- *29) 8 12 13 4 5 7 8
- *30) 8 12 13 4 5 6 13

Caminhos requeridos pelo Grafo(9)

- 31) 9 11 12 13 4 14 15 16
- 32) 9 11 12 13 4 5 7 9
- 33) 9 11 12 13 4 5 7 8 12
- 34) 9 11 12 13 4 5 6 13
- 35) 9 10 11 12 13 4 14 15 16
- *36) 9 10 11 12 13 4 5 7 9
- *37) 9 10 11 12 13 4 5 7 8 12
- *38) 9 10 11 12 13 4 5 6 13

Caminhos requeridos pelo Grafo(10)

- 39) 10 11 12 13 4 14 15 16
- *40) 10 11 12 13 4 5 7 9 11
- *41) 10 11 12 13 4 5 7 9 10
- *42) 10 11 12 13 4 5 7 8 12
- *43) 10 11 12 13 4 5 6

arquivo PUASSOC.TES

ASSOCIACOES REQUERIDAS PELOS CRITERIOS TODOS POT-USOS E POT-USOS/DU.

Associacoes requeridas pelo Grafo(1)

- 1) <1,(4,14),{ line, glob }>
- 2) <1,(9,11),{ line, glob }>
- 3) <1,(9,10),{ line, glob }>
- 4) <1,(7,8),{ line, glob }>
- 5) <1,(13,4),{ line, glob }>
- 6) <1,(5,6),{ line, glob }>
- 7) <1,(1,2),{ curln, line, glob }>

Associacoes requeridas pelo Grafo(2)

- 8) <2,(,),{ stat }>

Associacoes requeridas pelo Grafo(3)

- 9) <3,(4,14),{ curln, stat, done }>
- 10) <3,(9,11),{ curln, stat, done }>
- 11) <3,(9,10),{ curln, stat, done }>
- 12) <3,(7,8),{ curln, stat, done }>
- 13) <3,(13,4),{ curln, stat, done }>
- 14) <3,(13,4),{ curln, stat }>
- 15) <3,(13,4),{ curln, done }>
- 16) <3,(5,6),{ curln, stat, done }>

Associacoes requeridas pelo Grafo(5)

- 17) <5,(9,11),{ inline }>
- 18) <5,(9,10),{ inline }>
- 19) <5,(7,8),{ inline }>
- 20) <5,(4,14),{ inline }>
- 21) <5,(4,5),{ inline }>
- 22) <5,(5,6),{ inline }>

Associacoes requeridas pelo Grafo(6)

- 23) <6,(4,14),{ stat }>
- *24) <6,(9,11),{ stat }>
- *25) <6,(9,10),{ stat }>
- *26) <6,(12,13),{ stat }>
- 27) <6,(7,8),{ stat }>
- 28) <6,(5,6),{ stat }>

Associacoes requeridas pelo Grafo(8)

- 29) <8,(4,14),{ done }>
- 30) <8,(9,11),{ done }>
- 31) <8,(11,12),{ done }>
- 32) <8,(9,10),{ done }>
- *33) <8,(7,8),{ done }>
- *34) <8,(6,13),{ done }>
- *35) <8,(5,6),{ done }>

Associacoes requeridas pelo Grafo(9)

- 36) <9,(9,11),{ inline }>
- 37) <9,(4,14),{ inline }>
- 38) <9,(7,9),{ inline }>
- 39) <9,(8,12),{ inline }>
- 40) <9,(7,8),{ inline }>
- 41) <9,(6,13),{ inline }>
- 42) <9,(5,6),{ inline }>
- 43) <9,(9,10),{ inline }>

Associacoes requeridas pelo Grafo(10)

- 44) <10,(4,14),{ stat }>
- *45) <10,(9,11),{ stat }>
- *46) <10,(9,10),{ stat }>
- *47) <10,(8,12),{ stat }>
- *48) <10,(7,8),{ stat }>
- *49) <10,(5,6),{ stat }>

arquivo pathsnexec.tes

POTENCIAIS DU-CAMINHOS NAO EXECUTAVEIS

- 1) 1 3 4 14 15 16
- 8) 3 4 14 15 16
- 16) 5 7 9 10 11 12 13 4 5
- 18) 5 7 8 12 13 4 5
- 20) 5 6 13 4 5
- 22) 6 13 4 5 7 9 11 12 13
- 23) 6 13 4 5 7 9 10
- 24) 6 13 4 5 7 8 12 13
- 25) 6 13 4 5 6
- 27) 8 12 13 4 5 7 9 11 12
- 28) 8 12 13 4 5 7 9 10 11 12
- 29) 8 12 13 4 5 7 8
- 30) 8 12 13 4 5 6 13
- 36) 9 10 11 12 13 4 5 7 9
- 37) 9 10 11 12 13 4 5 7 8 12
- 38) 9 10 11 12 13 4 5 6 13
- 40) 10 11 12 13 4 5 7 9 11
- 41) 10 11 12 13 4 5 7 9 10
- 42) 10 11 12 13 4 5 7 8 12
- 43) 10 11 12 13 4 5 6

arquivo assocnexec.tes

ASSOCIACOES NAO EXECUTAVEIS

- 24) <6,(9,11),{ stat }>
- 25) <6,(9,10),{ stat }>
- 26) <6,(12,13),{ stat }>
- 33) <8,(7,8),{ done }>
- 34) <8,(6,13),{ done }>
- 35) <8,(5,6),{ done }>
- 45) <10,(9,11),{ stat }>
- 46) <10,(9,10),{ stat }>
- 47) <10,(8,12),{ stat }>
- 48) <10,(7,8),{ stat }>
- 49) <10,(5,6),{ stat }>

Todas as associações não executáveis foram determinadas pelo sistema. Existem ainda mais dois caminhos: 5 7 9 11 12 13 4 14 15 16 e 9 11 12 13 4 14 15 16 não executáveis que não foram determinados automaticamente, pois os valores para done e stat, estão indeterminados. O usuário poderá fornecê-los, e uma vez identificado como não executável

o segundo caminho, dele obtém-se um padrão que é utilizado para eliminar também o primeiro caminho.

.seleciona opção para o grafol -9

- a. Visualiza/Seleciona associação
- b. Visualiza/Seleciona pot-du-cam
- c. Mantem padroes
- d. Retorna para menu principal

Entre com a opção desejada

==> b

Mensagens -----

.seleciona o caminho

- 1) 9 11 12 13 4 14 15 16
- 2) 9 11 12 13 4 5 7 9
- 3) 9 11 12 13 4 5 7 8 12
- 4) 9 11 12 13 4 5 6 13
- 5) 9 10 11 12 13 4 14 15 16

Seleciona numero do potencial du-caminho ou retorna(R:r)

==> 5

. analisa predicados do caminho

Potencial du caminho: 9 10 11 12 13 4 14 15 16

Pred(4) = ((!done)&&(stat==2))

Para a caminho ser executavel predicado precisa valer false

Avalia predicado do no ou retorna(A:a//R:r)

==> a

Digite i para introdução de fatos associados ao no 4

Terminada a execução de comandos do no 4

Digite qualquer tecla para continuar ...

mostra resultado da avaliação do predicado

Variaveis do predicado

done = indeterminado

stat = indeterminado

O predicado nao pode ser avaliado

Digite qualquer tecla para continuar ...

Digite i para introducao de fatos associados ao no 4

i

. o usuário fornece o valor de done e stat no nó 4.

Introducao de fatos

Iniciarizar valor das variaveis no no 4

Digite o nome da variavel, seu valor e o tipo segundo tabela abaixo

a.real

b.inteiro

c.char

d.booleano

Tecle <ret> para encerrar

var: done tipo: b valor:0

var: stat tipo: b valor:2

. mostra resultado da avaliação

Variáveis do predicado

```
done = 0  
stat = 2
```

O predicado foi avaliado true

Digite qualquer tecla para continuar...

O caminho é' não executável

B.2 Rotina GETFNS

```
#define ponta_de_prova(num) if(printed_nodes % 10) (++printed_nodes; fprintf(p  
else  (++printed_nodes;  fprintf(path," %d",num);)  
  
int  errcount;  
int  nfiles;  
char  fname[10][50];  
int  fstat[10];  
void  getfns()  
  
/* Obtem nomes de arquivos de fname, procurando por duplicatas */  
  
/* 1 */      {  
    FILE * path = fopen("path.tes","w");  
    static int printed_nodes = 0;  
    int i,j;  
    ponta_de_prova(1);  
    /* 1 */  
    /* 1 */  
    /* 1 */  
    /* 1 */  
    /* 2 */  
    /* 2 */      {  
        ponta_de_prova(2);  
        printf("archive: too many file names");  
        ponta_de_prova(17);  
        fclose(path);  
        return;  
    /* 2 */  
    /* 2 */      }  
    ponta_de_prova(3);  
    /* 3 4 5 */      for(i=0;i<nfiles;i++)
```

```

/* 5 */
{
    ponta_de_prova(4);
    ponta_de_prova(5);
    getarg(i+3, fname[i]);
}

ponta_de_prova(4);
ponta_de_prova(6);
for(i=0;i<nfiles;i++)
{
    ponta_de_prova(7);
    ponta_de_prova(8);
    fstat[i] = 0;
}
ponta_de_prova(7);
ponta_de_prova(9);
for(i=0;i<nfiles-1;i++)
{
    ponta_de_prova(10);
    ponta_de_prova(11);
    for(j=(i+1);j<nfiles;j++)
    {
        ponta_de_prova(12);
        ponta_de_prova(13);
        if(equal(fname[i], fname[j]))
        {
            ponta_de_prova(14);
            puts(fname[i]);
            printf("duplicate file name");
            ponta_de_prova(17);
            fclose(path);
            return;
        }
        ponta_de_prova(15);
    }
    ponta_de_prova(12);
    ponta_de_prova(16);
}
ponta_de_prova(10);
ponta_de_prova(17);
fclose(path);
}
/* 17 */
}

```

Todos os elementos não executáveis são determinados através de interação com o usuário. A limitação para que sejam determinados automaticamente está na variável nfiles que possui um valor indeterminado resultante de uma chamada de função. Os arquivos pdupaths.tes e puassoc.tes permanecem inalterados.

CAMINHOS REQUERIDOS PELO CRITERIO TODOS POT-DU-CAMINHOS.

Caminhos requeridos pelo Grafo(1)

- 1) 1 3 4 6 7 9 10 17
- 2) 1 3 4 6 7 9 10 11 12 16 10
- 3) 1 3 4 6 7 9 10 11 12 13 15 12
- 4) 1 3 4 6 7 9 10 11 12 13 14 17
- 5) 1 3 4 6 7 8 7
- 6) 1 3 4 5 4
- 7) 1 2 17

Caminhos requeridos pelo Grafo(3)

- 8) 3 4 6
- 9) 3 4 5

Caminhos requeridos pelo Grafo(5)

- 10) 5 4 6
- 11) 5 4 5

Caminhos requeridos pelo Grafo(6)

- 12) 6 7 9
- 13) 6 7 8

Caminhos requeridos pelo Grafo(8)

- 14) 8 7 9 10 17
- 15) 8 7 9 10 11 12 16 10
- 16) 8 7 9 10 11 12 13 15 12
- 17) 8 7 9 10 11 12 13 14 17
- 18) 8 7 8

Caminhos requeridos pelo Grafo(9)

- 19) 9 10 17
- 20) 9 10 11 12 16
- 21) 9 10 11 12 13 15 12
- 22) 9 10 11 12 13 14 17

Caminhos requeridos pelo Grafo(11)

- 23) 11 12 16 10 17
- 24) 11 12 16 10 11
- 25) 11 12 13 15
- 26) 11 12 13 14 17

Caminhos requeridos pelo Grafo(15)

- 27) 15 12 16 10 17
- 28) 15 12 16 10 11
- 29) 15 12 13 15
- 30) 15 12 13 14 17

Caminhos requeridos pelo Grafo(16)

- 31) 16 10 17
- 32) 16 10 11 12 16
- 33) 16 10 11 12 13 15 12
- 34) 16 10 11 12 13 14 17

arquivo puassoc.tes

ASSOCIACOES REQUERIDAS PELOS CRITERIOS TODOS POT-USOS E POT-USOS/DU.

Associacoes requeridas pelo Grafo(1)

- 1) <1,(10,17),{ errcount, nfiles, fname, fstat }>
- 2) <1,(16,10),{ errcount, nfiles, fname, fstat }>
- 3) <1,(12,16),{ errcount, nfiles, fname, fstat }>
- 4) <1,(13,15),{ errcount, nfiles, fname, fstat }>
- 5) <1,(15,12),{ errcount, nfiles, fname, fstat }>
- 6) <1,(13,14),{ errcount, nfiles, fname, fstat }>
- 7) <1,(8,7),{ errcount, nfiles, fname }>
- 8) <1,(7,8),{ errcount, nfiles, fname, fstat }>
- 9) <1,(5,4),{ errcount, nfiles, fname, fstat }>
- 10) <1,(4,5),{ errcount, nfiles, fname, fstat }>
- 11) <1,(1,2),{ errcount, nfiles, fname, fstat }>

Associacoes requeridas pelo Grafo(3)

- 12) <3,(4,6),{ 1 }>
- 13) <3,(4,5),{ 1 }>

Associacoes requeridas pelo Grafo(5)

- 14) <5,(4,6),{ i }>
- 15) <5,(4,5),{ i }>

Associacoes requeridas pelo Grafo(6)

- 16) <6,(7,9),{ i }>
- 17) <6,(7,8),{ i }>

Associacoes requeridas pelo Grafo(8)

- 18) <8,(10,17),{ fstat }>
- 19) <8,(16,10),{ fstat }>
- 20) <8,(12,16),{ fstat }>
- 21) <8,(13,15),{ fstat }>
- 22) <8,(15,12),{ fstat }>
- 23) <8,(13,14),{ fstat }>
- 24) <8,(7,8),{ fstat, i }>

Associacoes requeridas pelo Grafo(9)

- 25) <9,(10,17),{ i }>
- 26) <9,(12,16),{ i }>
- 27) <9,(13,15),{ i }>
- 28) <9,(15,12),{ i }>
- 29) <9,(13,14),{ i }>

Associacoes requeridas pelo Grafo(11)

- 30) <11,(10,17),{ j }>
- 31) <11,(10,11),{ j }>
- 32) <11,(12,16),{ j }>
- 33) <11,(13,15),{ j }>
- 34) <11,(13,14),{ j }>

Associacoes requeridas pelo Grafo(15)

- 35) <15,(10,17),{ j }>
- 36) <15,(10,11),{ j }>
- 37) <15,(12,16),{ j }>
- 38) <15,(13,15),{ j }>
- 39) <15,(13,14),{ j }>

Associacoes requeridas pelo Grafo(16)

- 40) <16,(10,17),{ 1 }>
- 41) <16,(12,16),{ 1 }>
- 42) <16,(13,15),{ 1 }>
- 43) <16,(15,12),{ 1 }>
- 44) <16,(13,14),{ 1 }>

As associações requeridas pelo grafo(1), números de 2 a 6 são não executáveis. Elas exigem que o predicado do nó 10 seja verdadeiro em algum momento, para isso, basta que a função nargs retorne um valor maior que 2, para ter nfiles > 0; Isso é possível, o usuário poderá então fornecer um valor = 3, por exemplo para nfiles. Mas se nfiles > 0, a heurística de Frankl pode ser aplicada, pois a condição do laço do nó 7 será avaliada true, e fstat, variável das associações será sempre redefinida.

.seleciona o grafol a ser analisado

Apoio a geracao de casos de teste

Determinacao iterativa de nao executabilidade

Nos que possuem definicao de variaveis e grafol

no 1

no 3

no 5

no 6

no 8

no 9

no 11

no 15

no 16

Seleciona numero do no i ou retorna(R:r)

==> 1

.seleciona opcao para o grafol - 1

- a. Visualiza/Seleciona associacao
- b. Visualiza/Seleciona pot-du-cam
- c. Mantem padroes
- d. Retorna para menu principal

Entre com a opcao desejada

==> a

Mensagens -----

.seleciona associação

- 1) <1,(10,17),{ errcount, nfiles, fname, fstat }>
- 2) <1,(16,10),{ errcount, nfiles, fname, fstat }>
- 3) <1,(12,16),{ errcount, nfiles, fname, fstat }>
- 4) <1,(13,15),{ errcount, nfiles, fname, fstat }>
- 5) <1,(15,12),{ errcount, nfiles, fname, fstat }>
- 6) <1,(13,14),{ errcount, nfiles, fname, fstat }>
- 7) <1,(8,7),{ errcount, nfiles, fname }>
- 8) <1,(7,8),{ errcount, nfiles, fname, fstat }>
- 9) <1,(5,4),{ errcount, nfiles, fname, fstat }>
- 10) <1,(4,5),{ errcount, nfiles, fname, fstat }>
- 11) <1,(1,2),{ errcount, nfiles, fname, fstat }>

Seleciona numero da associacao ou retorna(R:r)

==> 2

.seleciona operação para a associação

Associacao selecionada: <1,(16,10),{ errcount, nfiles, fname, fstat }>

- a. Visualiza/Seleciona conjuntos estendidos a ciclo
- b. Elimina associacao
- c. Executa caso de teste
- d. Avalia caso de teste
- e. Retorna

Entre com a opção desejada:

==> a

Mensagens -----

.mostra conjuntos estendidos a ciclo

Conjuntos estendidos a ciclo para a associacao

<1,(16,10),{ errcount, nfiles, fname, fstat }> para o grafo 1.

1. 1 3 4 6 7 9 10 11 12 16

Seleciona numero do conjunto ou retorna(R|r)

==> 1

.analisar conjunto estendido a ciclo

Conjunto estendido a ciclo 1 3 4 6 7 9 10 11 12 16

Pred(7) = (i<nfiles)

Pred(7) = false

L4 = { }

Pred(7) = true

L7 = { 7 }

Para a associacao ser executada e necessario avaliar predicado false

Avalia predicado do no ou retorna(A:a//R:r)

==> a

Digite i para introducao de fatos associados ao no 7

Terminada a execucao de comandos do no 7

Digite qualquer tecla para continuar

i

. o usuário fornece um valor para nfiles

Introducao de fatos

Iniciarizar valor das variaveis no no 7

Digite o nome da variavel, seu valor e o tipo segundo tabela abaixo

a.real

b.inteiro

c.char

d.booleano

Tecle <ret> para encerrar

var:nfiles tipo:b valor:3

. mostra resultado da avaliação

Variáveis do predicado

i = 0

nfiles = 3

O predicado foi avaliado true

Digite qualquer tecla para continuar...

O associação é não executável

os caminhos associados são também eliminados; e da mesma maneira, interativamente determina-se a não executabilidade das outras associações.

arquivo pathsnexec.tes

POTENCIAIS DUT-CAMINHOS NAO EXECUTAVEIS

- 2) 1 3 4 6 7 9 10 11 12 16 10
- 3) 1 3 4 6 7 9 10 11 12 13 15 12
- 4) 1 3 4 6 7 9 10 11 12 13 14 17

arquivo assocnexec.tes

ASSOCIAÇOES NAO EXECUTAVEIS

- 2) <1,(16,10),{ errcount, nfiles, fname, fstat }>
- 3) <1,(12,16),{ errcount, nfiles, fname, fstat }>
- 4) <1,(13,15),{ errcount, nfiles, fname, fstat }>
- 5) <1,(15,12),{ errcount, nfiles, fname, fstat }>
- 6) <1,(13,14),{ errcount, nfiles, fname, fstat }>

As associações requeridas pelo grafo 11, números de 30 a 32, são não executáveis, pois se o predicado do nó 10 foi avaliado true, também o será o predicado do nó 12, pois esses são dependentes. O usuário poderá informar isso, avaliando o predicado do nó 12.

Introducao de fatos

Iniciar valor das variaveis no no 12

Digite o nome da variavel, seu valor e o tipo segundo tabela abaixo

- a.real
- b.inteiro
- c.char
- d.booleano

Tecle <ret> para encerrar

var: tipo: valor:

Como avaliar predicado do no 12

Pred(12) = (j<nfiles) = true

A associacao e' nao executavel

Finalmente, o caminho 1 3 4 6 7 8 7 é não executável, e poderá ser identificado se o usuário, analogamente, avaliar o predicado do nó 7. Esse predicado é necessariamente falso, pois é igual ao predicado do nó 4, que é falso para o caminho.

arquivo assocnexec.tes

ASSOCIACOES NAO EXECUTAVEIS

- 2) <1,(16,10),{ errcount, nfiles, fname, fstat }>
- 3) <1,(12,16),{ errcount, nfiles, fname, fstat }>
- 4) <1,(13,15),{ errcount, nfiles, fname, fstat }>
- 5) <1,(15,12),{ errcount, nfiles, fname, fstat }>
- 6) <1,(13,14),{ errcount, nfiles, fname, fstat }>
- 30) <11,(10,17),{ j }>
- 31) <11,(10,11),{ j }>
- 32) <11,(12,16),{ j }>

POTENCIAIS DU-CAMINHOS NAO EXECUTAVEIS

- 2) 1 3 4 6 7 9 10 11 12 16 10
- 3) 1 3 4 6 7 9 10 11 12 13 15 12
- 4) 1 3 4 6 7 9 10 11 12 13 14 17
- 5) 1 3 4 6 7 8 7
- 15) 8 7 9 10 11 12 16 10
- 20) 9 10 11 12 16
- 23) 11 12 16 10 17
- 24) 11 12 16 10 11
- 32) 16 10 11 12 16