

Teste de Software

Teste Funcional

Teste Estrutural

Teste Baseado em Erros (Análise de Mutantes)

Profa Rosana T. V. Braga

Material adaptado do material dos profs. Ellen Francine Barbosa e

José Carlos Maldonado

Técnica Funcional (Caixa Preta)

- Baseia-se na especificação do software para derivar os requisitos de teste
- Aborda o software de um ponto de vista macroscópico
- Envolve dois passos principais:
 - Identificar as funções que o software deve realizar (especificação dos requisitos)
 - Criar casos de teste capazes de checar se essas funções estão sendo executadas corretamente

Técnica Funcional

- Problema
 - Dificuldade em quantificar a atividade de teste: não se pode garantir que partes essenciais ou críticas do software foram executadas
 - Dificuldade de automatização
- Critérios da Técnica Funcional
 - Particionamento em Classes de Equivalência
 - Análise do Valor Limite
 - Grafo de Causa-Efeito

Técnica Funcional: Exemplo

- Particionamento em Classes de Equivalência
 - Divide o domínio de entrada do programa em classes de dados (classes de equivalências)
 - Os dados de teste são derivados a partir das classes de equivalência

Técnica Funcional: Exemplo

- Passos
 - Identificar classes de equivalência
 - Condições de entrada
 - Classes válidas e inválidas
 - Definir os casos de teste
 - Enumeram-se as classes de equivalência
 - Casos de teste para as classes válidas
 - Casos de teste para as classes inválidas

Técnica Funcional: Exemplo

➤ Especificação do programa *Identifier*

O programa deve determinar se um identificador é válido ou não. Um identificador válido deve começar com uma letra e conter apenas letras ou dígitos. Além disso, deve ter no mínimo um caractere e no máximo seis caracteres de comprimento.

➤ Exemplo

abc12 (válido);
cont*1 (inválido);

1soma (inválido);
a123456 (inválido)

Técnica Funcional: Exemplo

➤ Classes de equivalência

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho t do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caractere c é uma letra	Sim (3)	Não (4)
Só contém caracteres válidos	Sim (5)	Não (6)

➤ Exemplo de Conjunto de Casos de Teste

- $T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$
(1, 3, 5) (4) (6) (2)

Particionamento em Classes de Equivalência: Exemplo

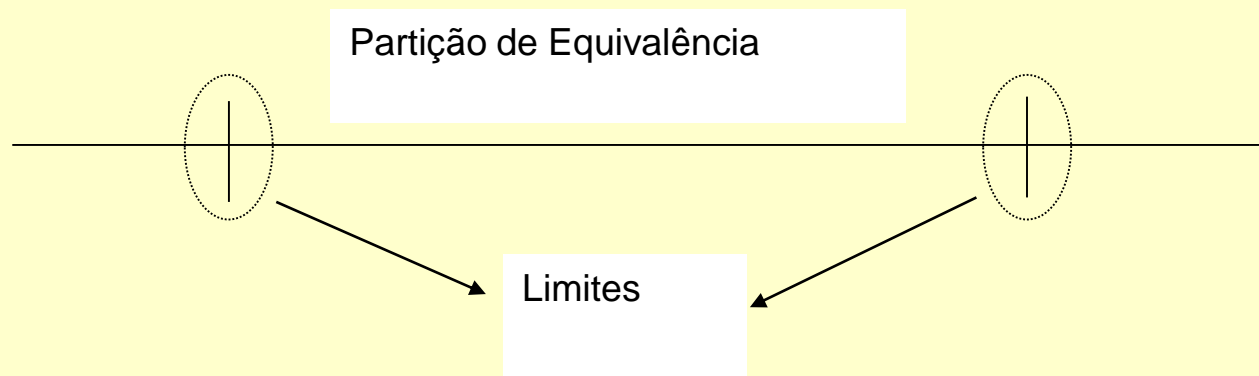
➤ Casos de Teste

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho da string	$1 \leq t \leq 20$ (1)	$t > 20$ $t < 1$ (2) (3)
Caractere procurado pertence à string	Sim (4)	Não (5)

Entrada	Saída	Classes Cobertas
34	entre com um inteiro entre 1 e 20	(2)
0	entre com um inteiro entre 1 e 20	(3)
3, abc, c	o caracter c aparece na posição 3	(1) e (4)
3, abc, k	o caracter k não ocorre na string fornecida	(5)

Análise do Valor Limite

- Complementa o Particionamento de Equivalência.
- Fonte propícia a erros – os **limites** de uma classe ou partição de equivalência.



Análise do Valor Limite: Exemplo

➤ Limites

➤ Tamanho da string.

- Os valores inteiros 0, 1, 20 e 21.

➤ Caracter a ser procurado.

- Encontrar o caracter na primeira e na última posição da cadeia de caracteres.

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho da string	$1 \leq t \leq 20$ (1)	$t > 20$ (2) $t < 1$ (3)
Caractere procurado pertence à string	Sim (4)	Não (5)

Entrada	Saída	Classes Cobertas
21	entre com um inteiro entre 1 e 20	(2)
0	entre com um inteiro entre 1 e 20	(3)
1 , a, a	o caracter a aparece na posição 1	(1) e (4)
1 , a, x	o caracter x não ocorre na string fornecida	(5)
20 , abcdefghijklmnopqrst, a	o caracter a aparece na posição 1	(1) e (4)
20 , abcdefghijklmnopqrst, t	o caracter t aparece na posição 20	(1) e (4)

Técnica Estrutural (Caixa Branca)

- Baseada no conhecimento da estrutura interna (implementação) do programa
- Teste dos detalhes procedimentais
- A maioria dos critérios dessa técnica utilizam uma representação de programa conhecida como grafo de programa ou grafo de fluxo de controle

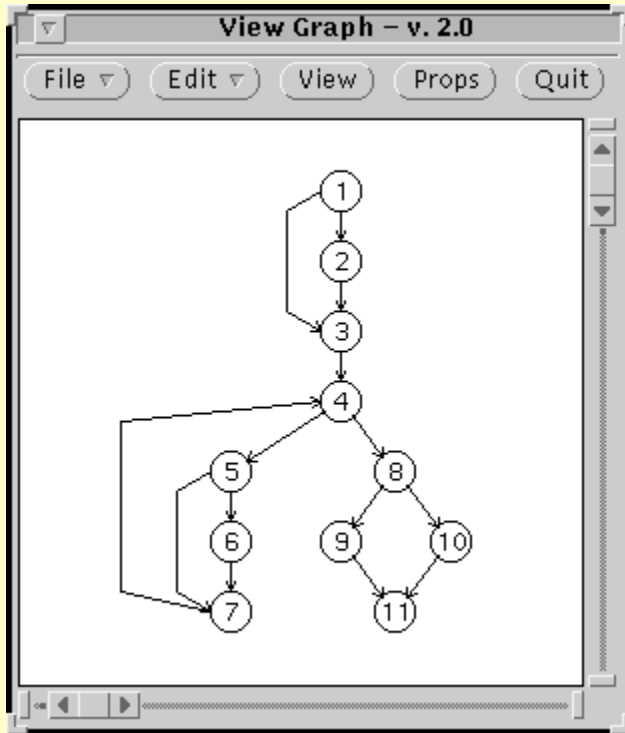
Técnica Estrutural

- Grafo de Programa
 - Nós: blocos “indivisíveis”
 - Não existe desvio para o meio do bloco
 - Uma vez que o primeiro comando do bloco é executado, os demais comandos são executados seqüencialmente
 - Arestas ou Arcos: representam o fluxo de controle entre os nós

Identifcier.c (função *main*)

```
/* 01 */ {
/* 01 */     char  achar;
/* 01 */     int   length, valid_id;
/* 01 */     length = 0;
/* 01 */     printf ("Identificador: ");
/* 01 */     achar = fgetc (stdin);
/* 01 */     valid_id = valid_s(achar);
/* 01 */     if (valid_id)
/* 02 */         length = 1;
/* 03 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!(valid_f(achar)))
/* 06 */             valid_id = 0;
/* 07 */         length++;
/* 07 */         achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (length < 6) )
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }
```

Técnica Estrutural



Grafo de Programa

- Detalhes considerados
 - nó
 - arco
 - caminho
 - simples (2,3,4,5,6,7)
 - completo (1,2,3,4,5,7,4,8,9,11)
 - fluxo de controle

Grafo de Programa do *identifier*
Gerado pela *View-Graph*

Identifier.c (função *main*)

```
/* 01 */ {
/* 01 */     char  achar;
/* 01 */     int   length, valid_id;
/* 01 */     length = 0;
/* 01 */     printf ("Identificador: ");
/* 01 */     achar = fgetc (stdin);
/* 01 */     valid_id = valid_s(achar);
/* 01 */     if (valid_id)
/* 02 */         length = 1;
/* 03 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!(valid_f(achar)))
/* 06 */             valid_id = 0;
/* 07 */         length++;
/* 07 */         achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (length < 6) )
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }
```

Caminho
Não-Executável

Técnica Estrutural

- Critérios da Técnica Estrutural
 - Baseados em Fluxo de Controle
 - Todos-Nós, Todas-Arestas e Todos-Caminhos
 - Baseados em Fluxo de Dados
 - Critérios de Rapps e Weyuker
 - Todas-Defs, Todos-Usos, Todos-P-Usos e outros
 - Critérios Potenciais-Usos (Maldonado)
 - Todos-Potenciais-Usos, Todos-Potenciais-Usos/DU e outros
 - Baseados em Complexidade
 - Critério de McCabe

Técnica Estrutural

➤ Critérios Baseados em Fluxo de Controle

➤ Todos-Nós

1,2,3,4,5,6,7,8,9,10,11

➤ Todos-Arcos

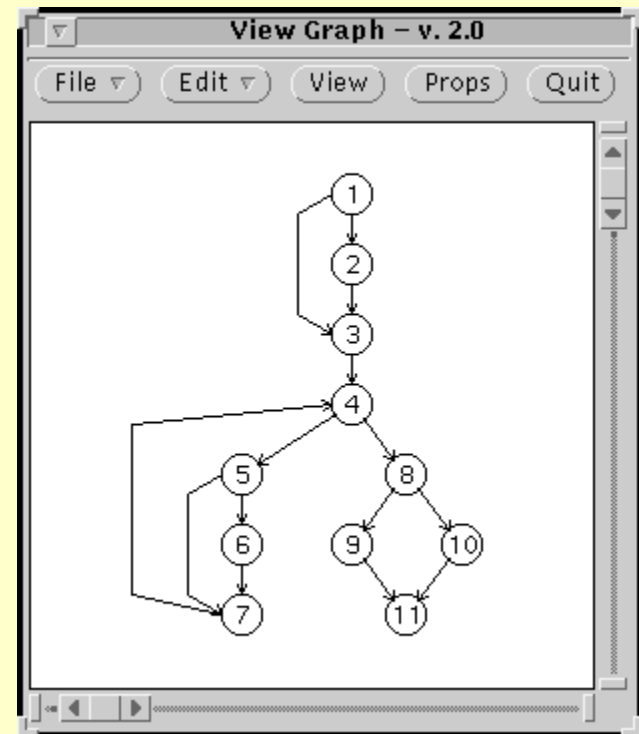
➤ arcos primitivos

$\langle 1,2 \rangle, \langle 1,3 \rangle, \langle 5,6 \rangle, \langle 5,7 \rangle,$

$\langle 8,9 \rangle, \langle 8,10 \rangle$

➤ Todos-Caminhos

Arcos que correspondem aos caminhos essenciais do modelo



Grafo de Programa do *identifier*
Gerado pela *View-Graph*

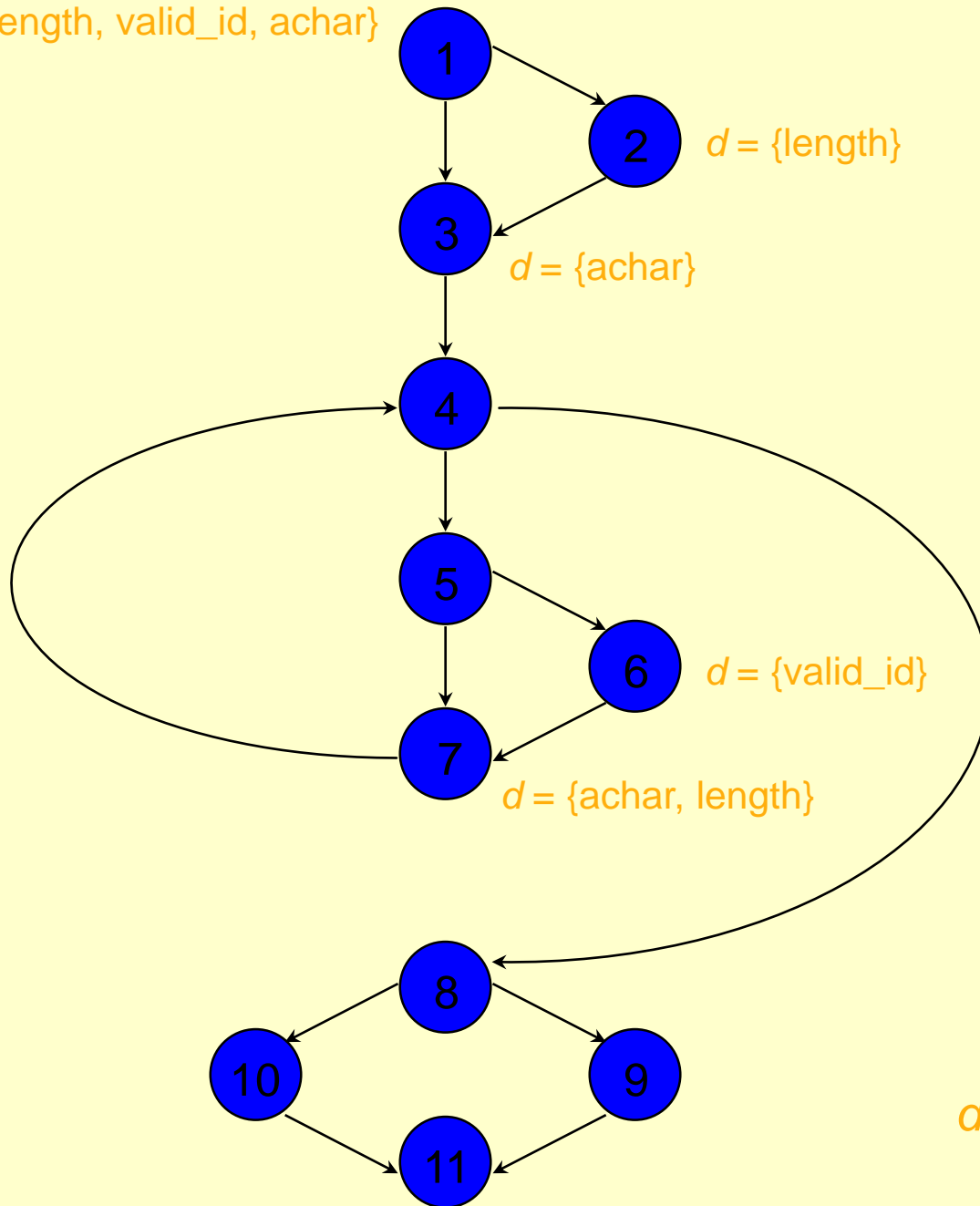
Técnica Estrutural

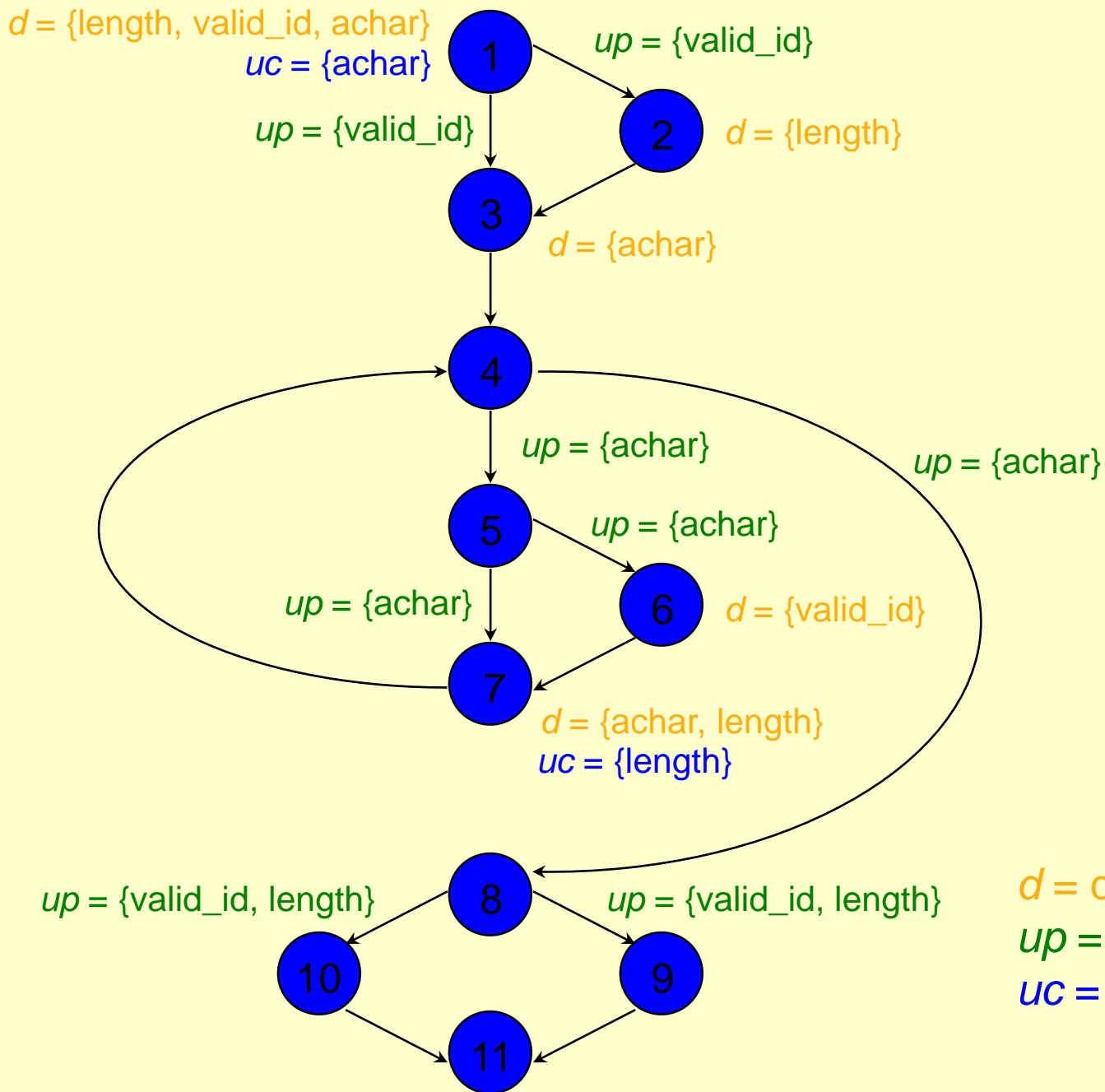
- Critérios Baseados em Fluxo de Dados
 - Rapps e Weyuker

Grafo Def-Uso: Grafo de Programa + Definição e Uso de Variáveis

- Definição
 - Atribuição de um valor a uma variável ($a = 1$)
- Uso
 - Predicativo: a variável é utilizada em uma condição
if ($a > 0$)
 - Computacional: a variável é utilizada em uma computação
 $b = a + 1$

$d = \{\text{length, valid_id, achar}\}$





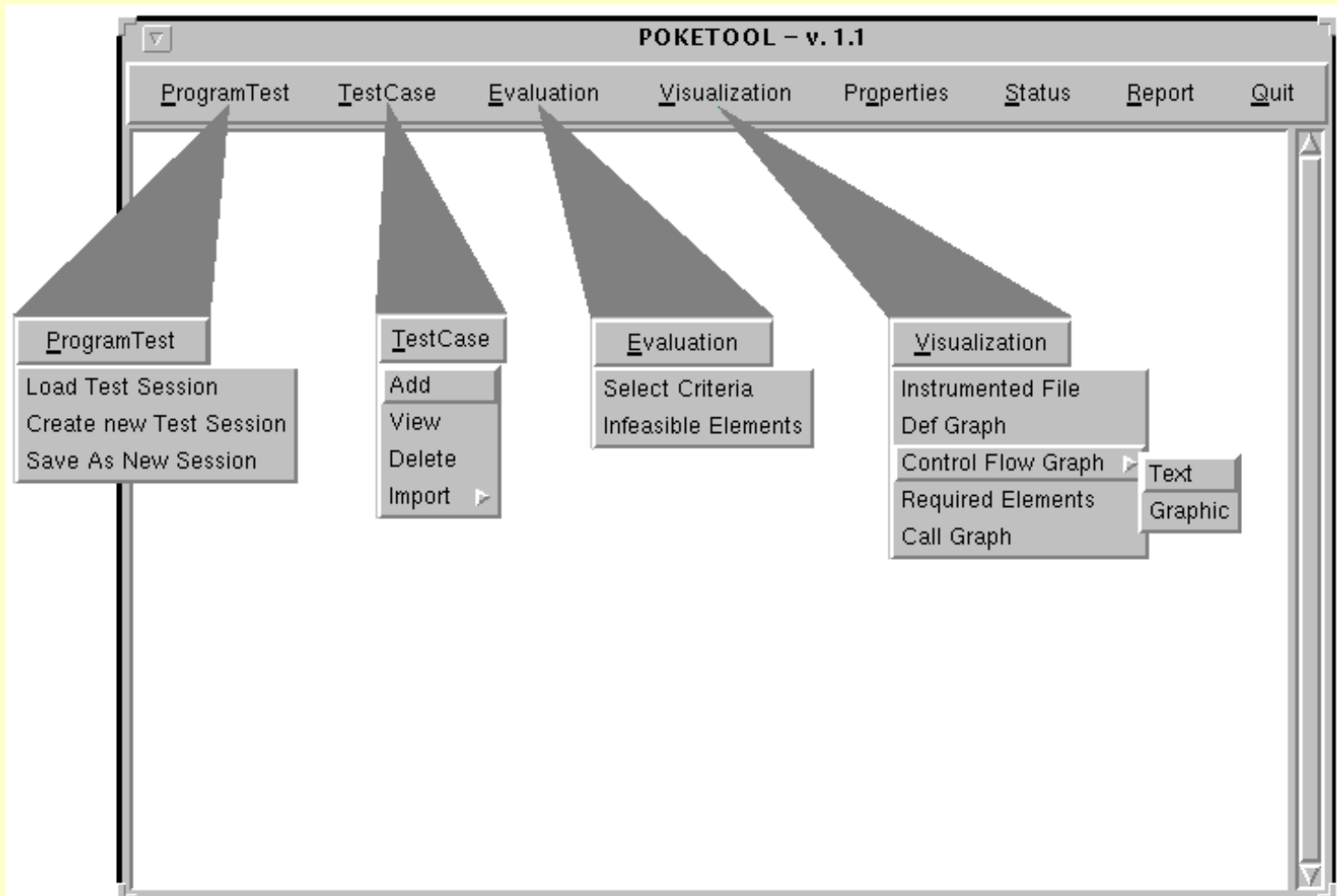
$d =$ definição
 $up =$ uso predicativo
 $uc =$ uso computacional

Técnica Estrutural

- Ferramenta *PokeTool*
 - Critérios Potenciais-Usos
 - Critérios de Rapps e Weyuker
 - Outros Critérios Estruturais
 - Todos-Nós, Todos-Arcos
 - Linguagem C
 - Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Geração de relatórios

Técnica Estrutural

➤ *PokeTool*: Interface Gráfica



Técnica Estrutural

- *PokeTool*: Criando uma Sessão de Teste

Create New Test Session

Directory: /export/home1/es/ellen/escola/poketool

Test Session Name: l

Source Program: i.c

Included Files:

Used Defines:

Functions: main

Compilation Command: gcc <source> -o <exec> -w

Criteria:

- All Node
- All Edges
- All Potential Uses
- All Potential Uses/DU
- All Potential DU-paths

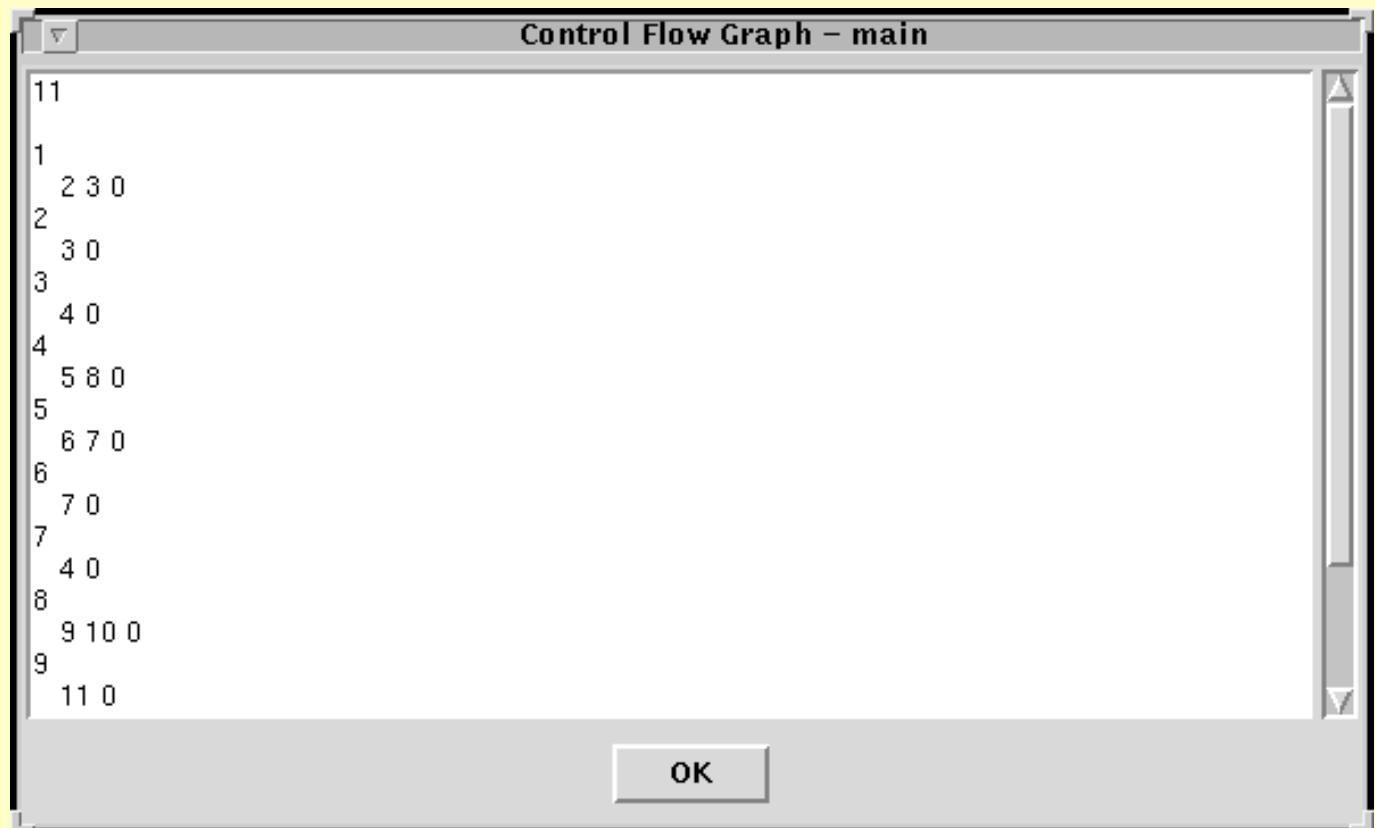
Type:

- test
- research

Confirm **Cancel**

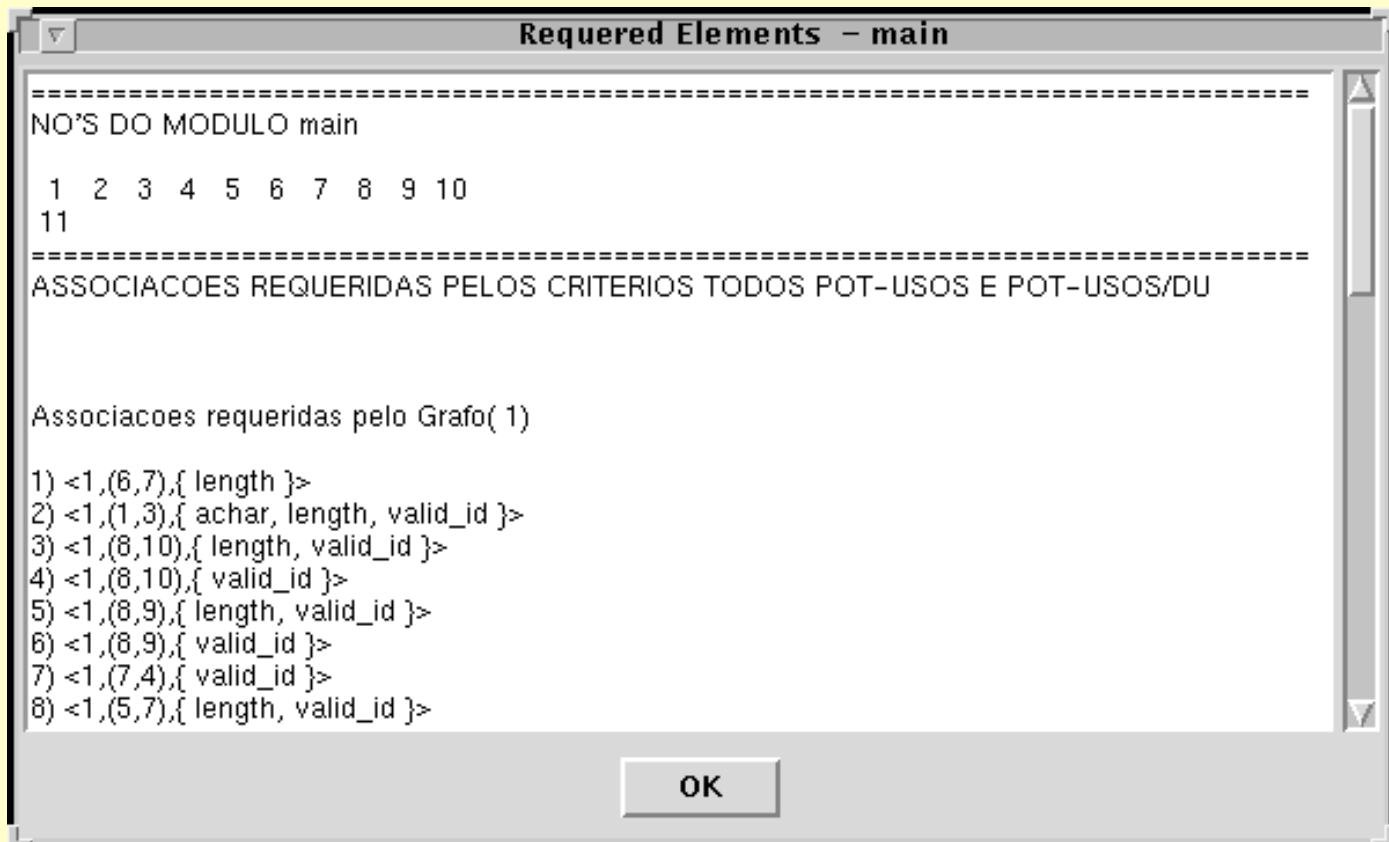
Técnica Estrutural

- *PokeTool*: Grafo de Programa



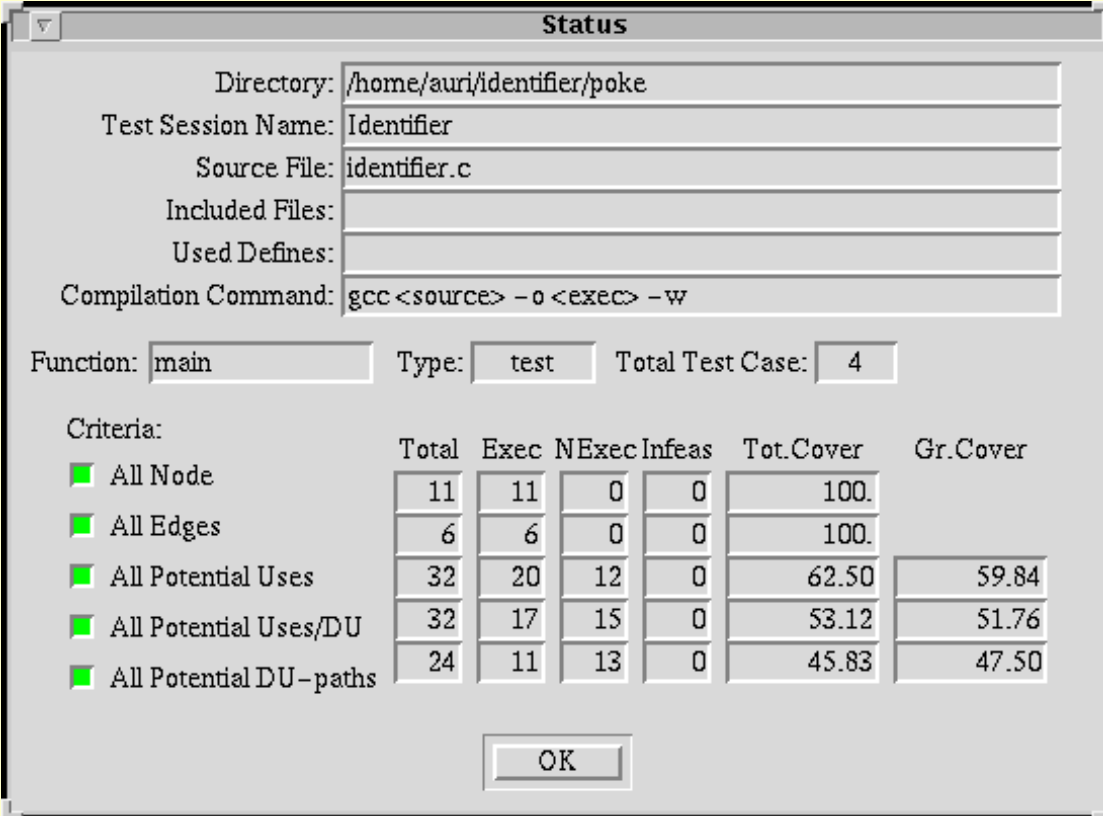
Técnica Estrutural

➤ *PokeTool*: Elementos Requeridos



Técnica Estrutural

➤ Status após T_0



The screenshot shows a 'Status' dialog box with the following fields:

- Directory: /home/auri/identifier/poke
- Test Session Name: Identifier
- Source File: identifier.c
- Included Files:
- Used Defines:
- Compilation Command: gcc <source> -o <exec> -w
- Function: main
- Type: test
- Total Test Case: 4

Criteria:

	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
<input checked="" type="checkbox"/> All Node	11	11	0	0	100.	
<input checked="" type="checkbox"/> All Edges	6	6	0	0	100.	
<input checked="" type="checkbox"/> All Potential Uses	32	20	12	0	62.50	59.84
<input checked="" type="checkbox"/> All Potential Uses/DU	32	17	15	0	53.12	51.76
<input checked="" type="checkbox"/> All Potential DU-paths	24	11	13	0	45.83	47.50

OK

➤ $T_0 = \{(a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido})\}$

Técnica Estrutural

➤ *PokeTool*: Relatórios de Teste

```
Report - main
ASSOCIACOES DO CRITERIO TODOS POT-USOS executadas:
<1,(1,3),{ achar, valid_id }>
<1,(8,10),{ valid_id }>
<1,(8,9),{ valid_id }>
<1,(7,4),{ valid_id }>
<1,(5,7),{ length, valid_id }>
<1,(5,7),{ valid_id }>
<1,(5,6),{ valid_id }>
<1,(2,3),{ achar, valid_id }>
<1,(1,2),{ achar, length, valid_id }>
<2,(5,7),{ length }>
<2,(6,7),{ length }>
<2,(5,6),{ length }>
<3,(5,7),{ achar }>
<3,(6,7),{ achar }>
<3,(5,6),{ achar }>
<6,(8,10),{ valid_id }>
<6,(5,7),{ valid_id }>
<7,(8,10),{ achar, length }>
<7,(8,9),{ achar, length }>
<7,(5,7),{ achar, length }>

Cobertura Total = 62.500000

Media da Cobertura dos Grafo(i) = 59.846153

OK
```

```
Report - main
ASSOCIACOES DO CRITERIO TODOS POT-USOS nao executadas:
<1,(6,7),{ length }>
<1,(8,10),{ length, valid_id }>
<1,(8,9),{ length, valid_id }>
<1,(5,6),{ length, valid_id }>
<2,(8,10),{ length }>
<2,(8,9),{ length }>
<3,(8,10),{ achar }>
<3,(8,9),{ achar }>
<6,(8,9),{ valid_id }>
<6,(5,6),{ valid_id }>
<7,(6,7),{ achar, length }>
<7,(5,6),{ achar, length }>

Cobertura Total = 62.500000

Media da Cobertura dos Grafo(i) = 59.846153

OK
```

Técnica Estrutural

➤ Status após T_1 (a) e T_2 (b)

Directory: /home/auri/identifler/poke
 Test Session Name: Identifier
 Source File: identifler.c
 Included Files:
 Used Defines:
 Compilation Command: gcc <source> -o <exec> -w

Function: main Type: test Total Test Case: 7

Criteria:

	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
All Node	11	11	0	0	100.	
All Edges	6	6	0	0	100.	
All Potential Uses	32	26	6	0	81.25	76.46
All Potential Uses/DU	32	26	6	0	81.25	76.46
All Potential DU-paths	24	18	6	0	75.00	75.00

OK

(a)

Directory: /home/auri/identifler/poke
 Test Session Name: Identifier
 Source File: identifler.c
 Included Files:
 Used Defines:
 Compilation Command: gcc <source> -o <exec> -w

Function: main Type: test Total Test Case: 8

Criteria:

	Total	Exec	NExec	Infeas	Tot.Cover	Gr.Cover
All Node	11	11	0	0	100.	
All Edges	6	6	0	0	100.	
All Potential Uses	32	29	3	0	90.62	89.46
All Potential Uses/DU	32	29	3	0	90.62	89.46
All Potential DU-paths	24	20	4	0	83.33	85.00

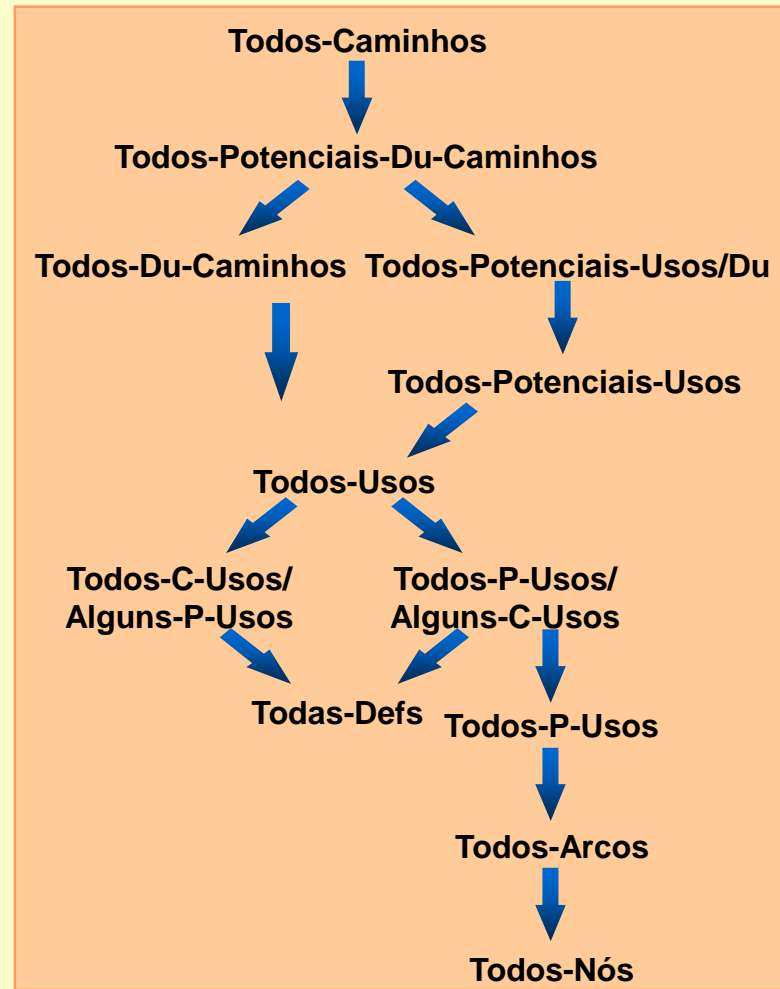
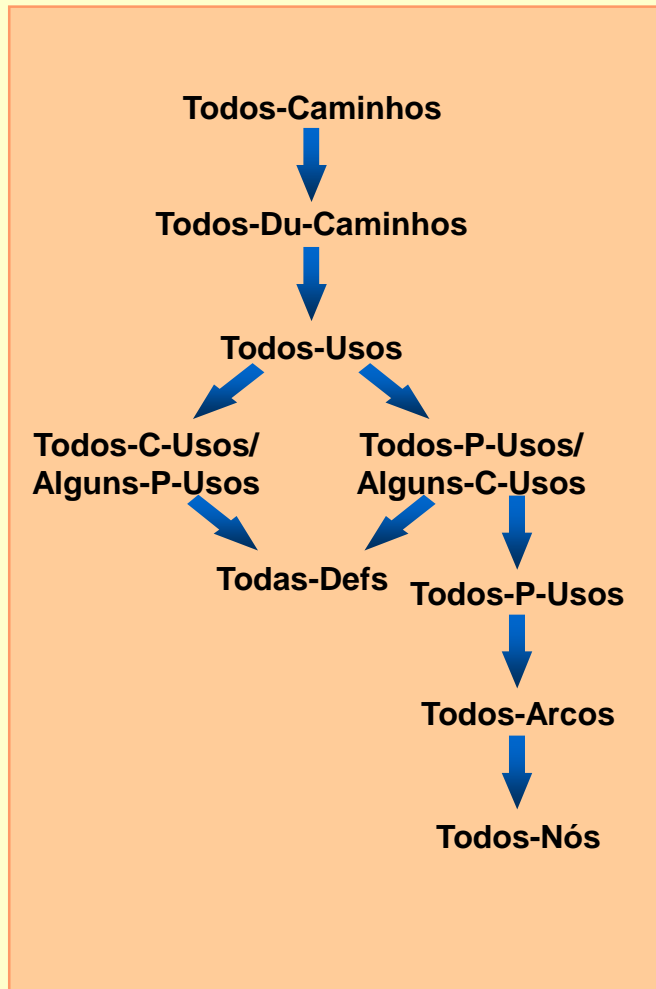
OK

(b)

- $T_1 = T_0 \cup \{(1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido})\}$
- $T_2 = T_1 \cup \{(\#-\%, \text{Inválido})\}$

Técnica Estrutural

➤ Hierarquia entre Critérios Estruturais



Técnica Baseada em Erros

- Os requisitos de teste são derivados a partir dos erros mais freqüentes cometidos durante o processo de desenvolvimento do software
- Critérios da Técnica Baseada em Erros
 - Semeadura de Erros
 - Teste de Mutação
 - Análise de Mutantes (unidade)
 - Mutação de Interface (integração)

Análise de Mutantes

➤ Hipótese do Programador Competente

Programadores experientes escrevem programas corretos ou muito próximos do correto.

➤ Efeito de Acoplamento

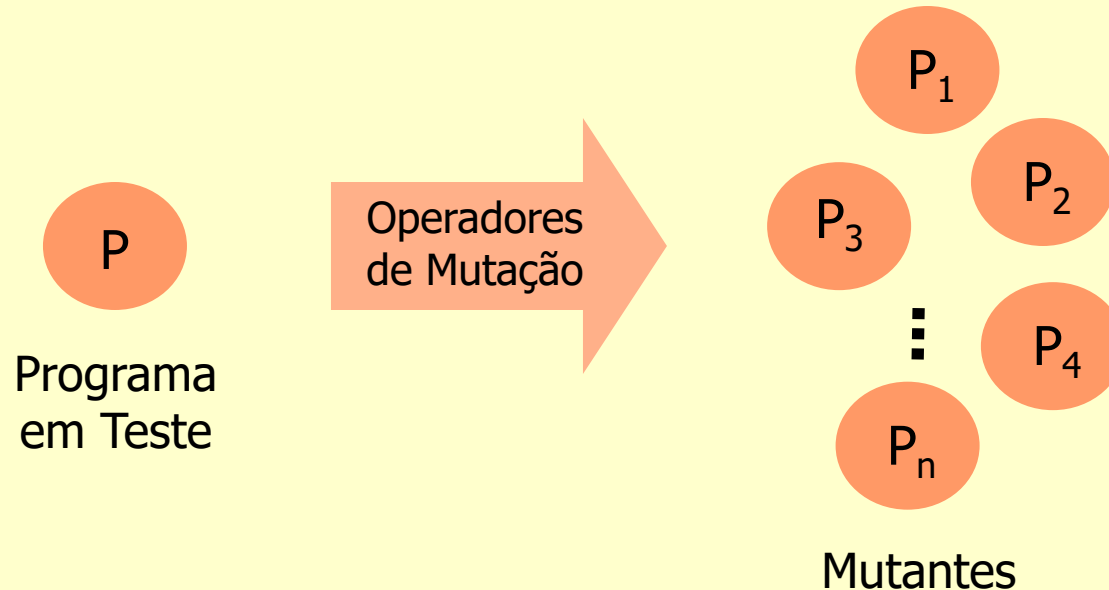
Casos de teste capazes de revelar erros simples são tão sensíveis que, implicitamente, também são capazes de revelar erros mais complexos.

Análise de Mutantes

➤ Passos da Análise de Mutantes

1- Geração de Mutantes

*Para modelar os desvios sintáticos mais comuns, **operadores de mutação** são aplicados a um programa, transformando-o em programas similares: **mutantes**.*



Análise de Mutantes

- Seleção dos operadores de mutação
 - Abrangente
 - Capaz de modelar a maior parte dos erros
 - Pequena cardinalidade
 - Problemas de custo
 - Quanto maior o número de operadores utilizados, maior o número de mutantes gerados

Análise de Mutantes

➤ Exemplo de Mutantes

Mutante Gerado pelo Operador OLAN

if (valid_id * (length >= 1) && (length < 6))
 printf ("Valido\n");
else
 printf ("Invalido\n");

OLAN: Troca
operador lógico por
operador aritmético

Mutante Gerado pelo Operador ORRN

if (valid_id && (length >= 1) && (length <= 6))
 printf ("Valido\n");
else
 printf ("Invalido\n");

ORRN: troca de
operador
relacional

Análise de Mutantes

➤ Passos da Análise de Mutantes

2 - Execução do Programa

- Execução do programa com os casos de teste

3 - Execução dos Mutantes

- Execução dos mutantes com os casos de teste
 - Mutante morto
 - Mutante vivo

4 - Análise dos Mutantes Vivos

- Mutante equivalente
- Inclusão de novos casos de teste

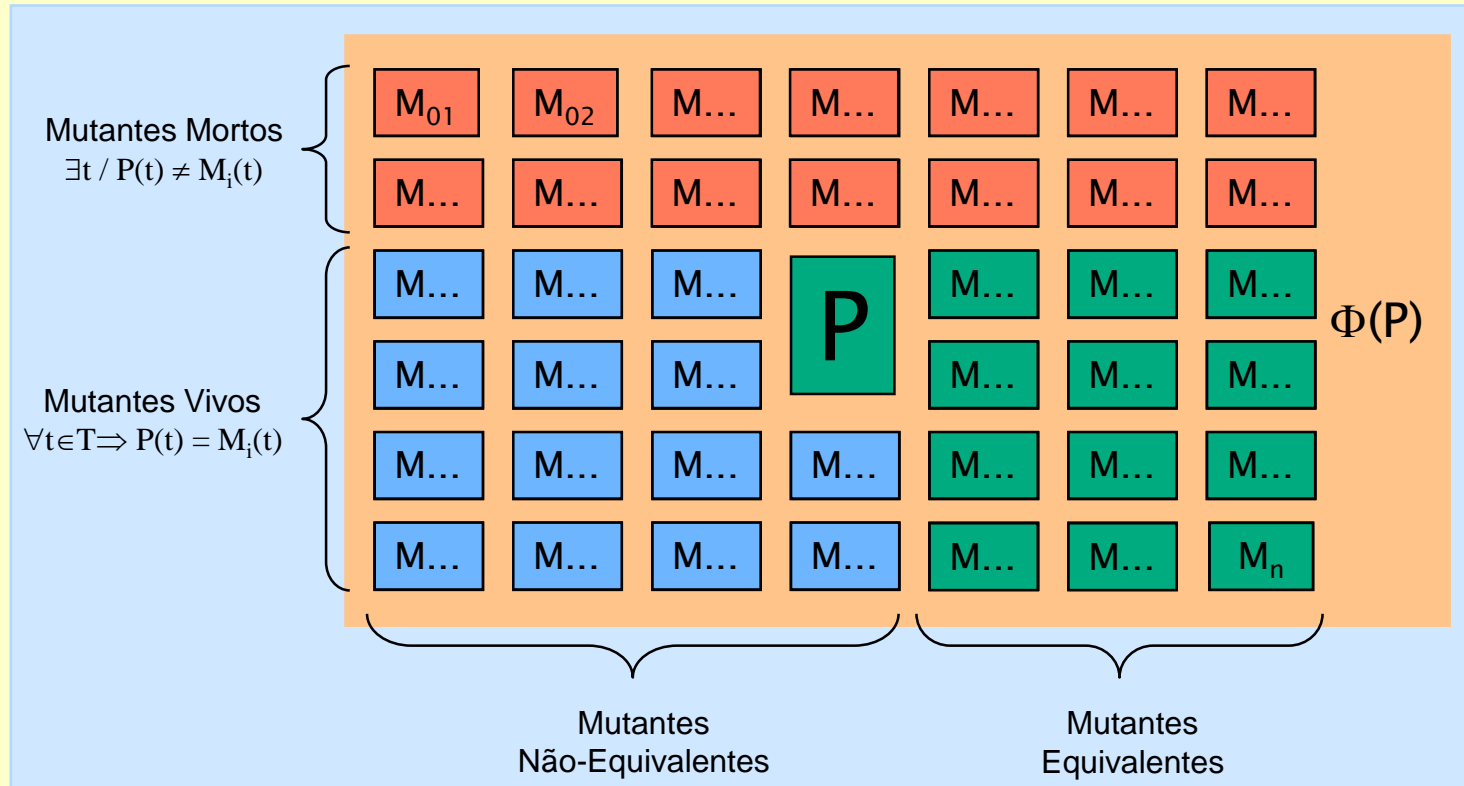
Escore de mutação:

$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

The diagram illustrates the mutation score formula with callouts to its components:

- ms(P,T)**: Mutation score
- DM(P,T)**: Dead mutant
- M(P) - EM(P)**: equivalent mutant

Análise de Mutantes



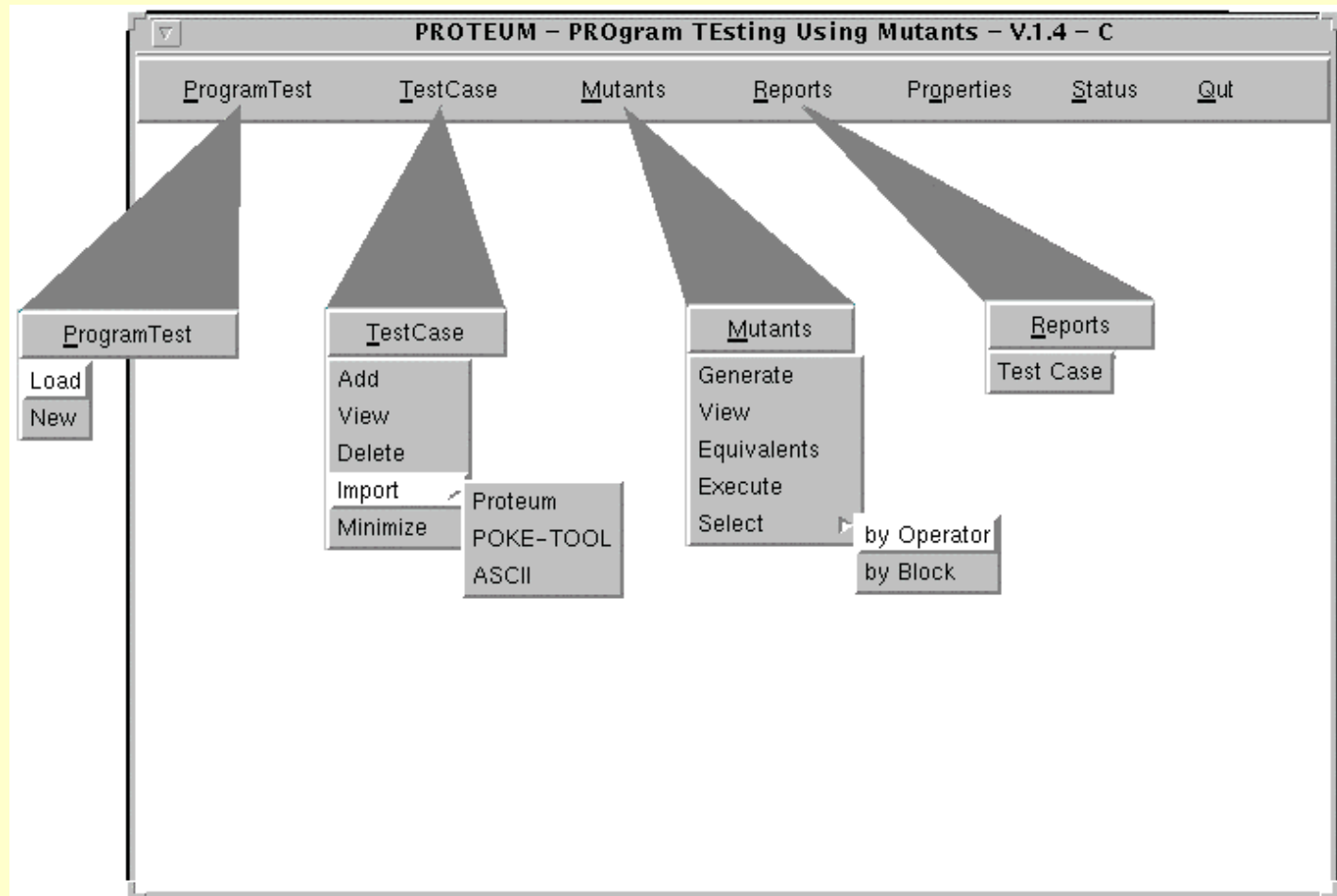
$$ms(P,T) = \frac{DM(P,T)}{M(P) - EM(P)}$$

Análise de Mutantes

- Ferramenta *Proteum*
 - Critério Análise de Mutantes
 - Linguagem C
 - Outras Características
 - Importação de casos de teste
 - Inserção e remoção de casos de teste dinamicamente
 - Casos de teste podem ser habilitados ou desabilitados
 - Seleção dos operadores a serem utilizados
 - 71 operadores: comandos, operadores, variáveis e constantes
 - Geração de relatórios
 - Scripts possibilitam a condução de uma sessão de teste de modo programado

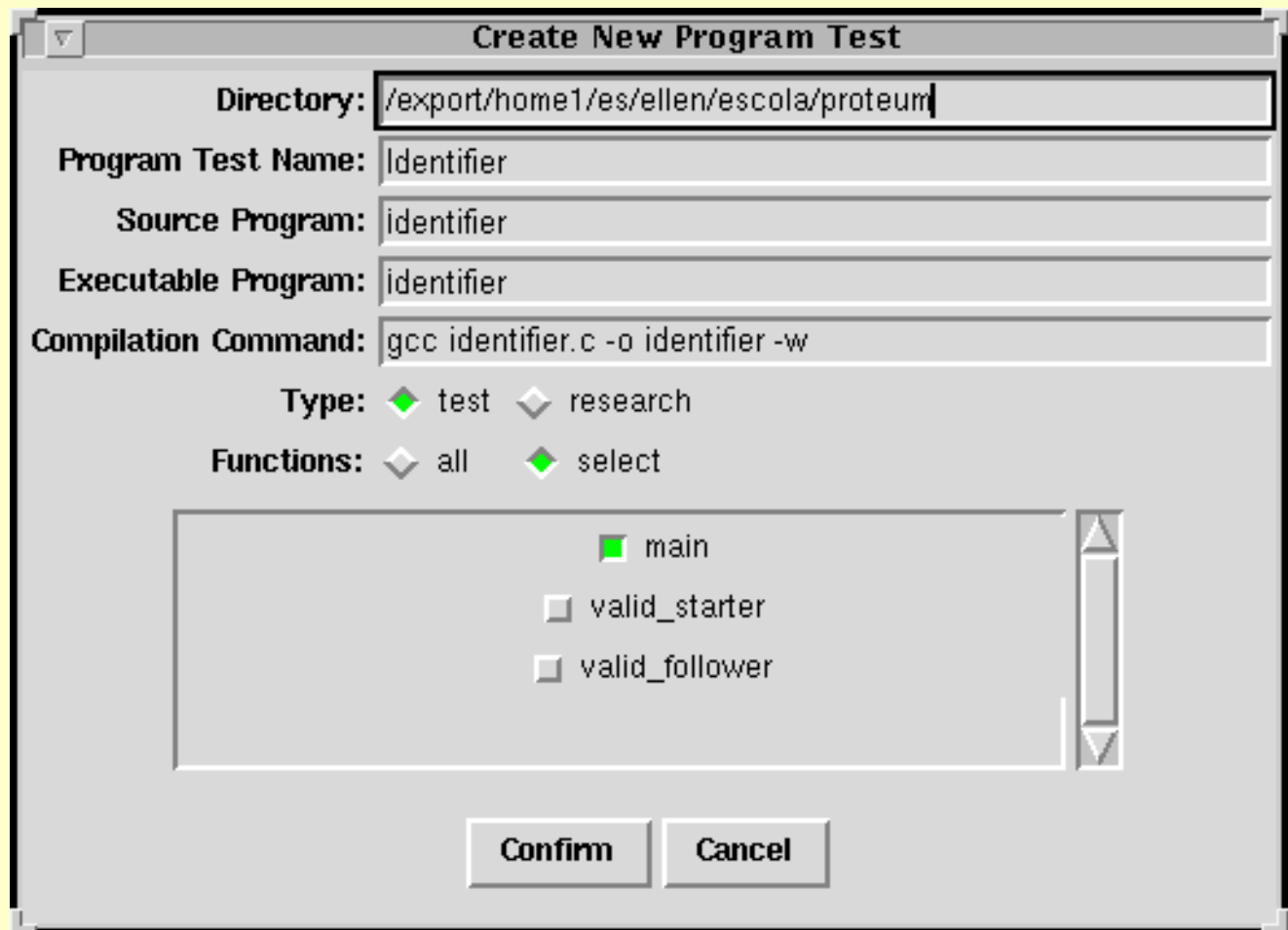
Análise de Mutantes

➤ *Proteum*: Interface Gráfica



Análise de Mutantes

- *Proteum*: Criando uma Sessão de Teste



The image shows a dialog box titled "Create New Program Test" with the following fields and options:

- Directory:** /export/home1/es/ellen/escola/proteum
- Program Test Name:** Identifier
- Source Program:** identifier
- Executable Program:** identifier
- Compilation Command:** gcc identifier.c -o identifier -w
- Type:** test research
- Functions:** all select

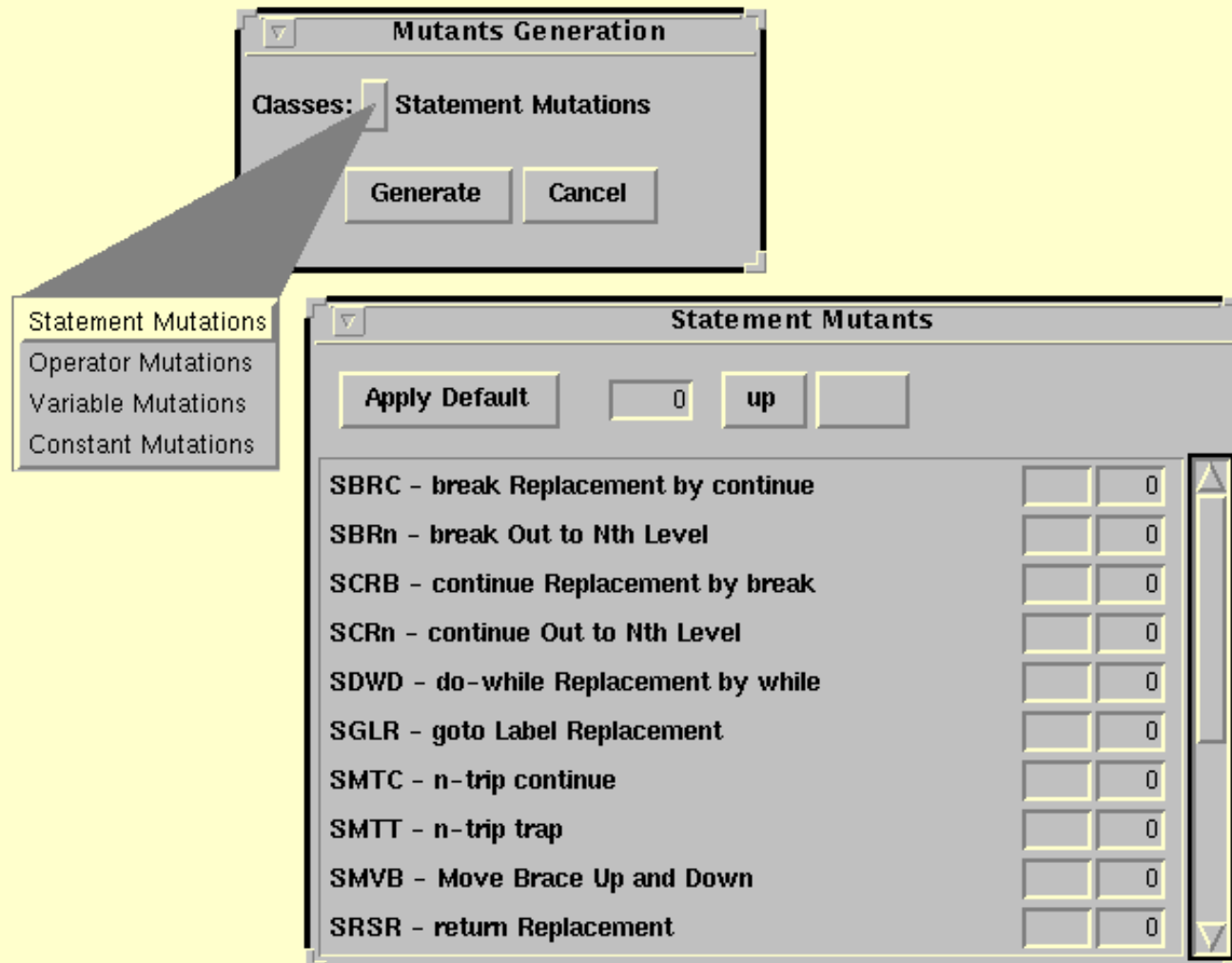
A list of functions to test is shown in a scrollable area:

- main
- valid_starter
- valid_follower

At the bottom, there are two buttons: **Confirm** and **Cancel**.

Análise de Mutantes

➤ *Proteum*: Gerando Mutantes



Análise de Mutantes

- Status após T_0 (a) e T_2 (b)

Directory:	/home/auri/identifier/tepeum		
Program Test Name:	Identifier		
Source Program:	identifier		
Executable Program:	identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	4
Total Mutants:	933	Live Mutants:	403
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	0	MUTATION SCORE:	0.568

(a)

Directory:	/home/auri/identifier/tepeum		
Program Test Name:	Identifier		
Source Program:	identifier		
Executable Program:	identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	8
Total Mutants:	933	Live Mutants:	371
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	0	MUTATION SCORE:	0.602

(b)

- $T_0 = \{ (a1, \text{Válido}), (2B3, \text{Inválido}), (Z-12, \text{Inválido}), (A1b2C3d, \text{Inválido}) \}$
- $T_1 = T_0 \cup \{ (1\#, \text{Inválido}), (\%, \text{Inválido}), (c, \text{Válido}) \}$
- $T_2 = T_1 \cup \{ (\#-\%, \text{Inválido}) \}$

Análise de Mutantes

➤ *Proteum*: Visualização de Mutantes

The screenshot displays the Proteum 'View Mutants' window. At the top, there are controls for the mutant: a 'Mutant' field with '0', 'up', and 'dw' buttons; a 'Type to Show' section with checkboxes for 'Alive', 'Dead', 'Anomalous', 'Equivalent', and 'Inactive'; a 'Status' field showing 'Alive; Active' and an 'Equivalent' checkbox; and an 'Operator' field showing 'Cccr - Constant for Constant Replacement'.

Below these controls are two side-by-side code editors. The left editor, titled 'Original Program', contains the following C code:

```
main ()
{
  char  achar;
  int   length, valid_id;

  >> length = 0;

  printf ("Digite um possivel identificador Silly Pascal\n");
  printf ("seguido por <ENTER>: ");

  achar = fgetc (&__iob[0] );
  valid_id = valid_starter (achar);

  if (valid_id)
    length = 1;
  achar = fgetc (&__iob[0] );

  while (achar != '\n') {
    if (!(valid_follower (achar))){
      valid_id = 0;
    }
    length++;
    achar = fgetc (&__iob[0] );
  }
```

The right editor, titled 'Mutant Program', shows the same code with a mutation: the initial assignment of 'length' is changed from 0 to 1, indicated by red '>>' markers:

```
main ()
{
  char  achar;
  >> int   length, valid_id;

  (length = 1);

  printf ("Digite um possivel identificador Silly Pascal\n");
  printf ("seguido por <ENTER>: ");

  achar = fgetc (&__iob[0] );
  valid_id = valid_starter (achar);

  if (valid_id)
    length = 1;
  achar = fgetc (&__iob[0] );

  while (achar != '\n') {
    if (!(valid_follower (achar))){
      valid_id = 0;
    }
    length++;
    achar = fgetc (&__iob[0] );
  }
```

Análise de Mutantes

➤ Status após T_3 (a) e T_4 (b)

Directory:	/home/auri/identifer/proteum		
Program Test Name:	identifer		
Source Program:	identifer		
Executable Program:	identifer		
Compilation Command:	gcc identifer.c -o identifer -w		
Type:	Test	Test Cases:	21
Total Mutants:	933	Live Mutants:	64
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	78	MUTATION SCORE:	0.925

(a)

Directory:	/home/auri/identifer/proteum		
Program Test Name:	identifer		
Source Program:	identifer		
Executable Program:	identifer		
Compilation Command:	gcc identifer.c -o identifer -w		
Type:	Test	Test Cases:	25
Total Mutants:	933	Live Mutants:	2
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	136	MUTATION SCORE:	0.997

(b)

- $T_3 = T_2 \cup \{(zzz, \text{Válido}), (aA, \text{Válido}), (A1234, \text{Válido}), (ZZZ, \text{Válido}), (AAA, \text{Válido}), (aa09, \text{Válido}), ([, \text{Inválido}), (\{, \text{Inválido}), (x/, \text{Inválido}), (x:, \text{Inválido}), (x18, \text{Válido}), (x[, \text{Inválido}), (x\{\{, \text{Inválido})\}$
- $T_4 = T_3 \cup \{(@, \text{Inválido}), (` , \text{Inválido}), (x@, \text{Inválido}), (x` , \text{Inválido})\}$

Análise de Mutantes

➤ Mutantes Vivos

Mutante Gerado pelo Operador VTWD

```
if (valid_id && (length >= 1) && (PRED(length) < 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

Mutante Gerado pelo Operador ORRN

```
if (valid_id && (length >= 1) && (length <= 6) )  
    printf ("Valido\n");  
else  
    printf ("Invalido\n");
```

$t = \{(ABCDEF, \text{Válido})\}$

Saída obtida = Inválido

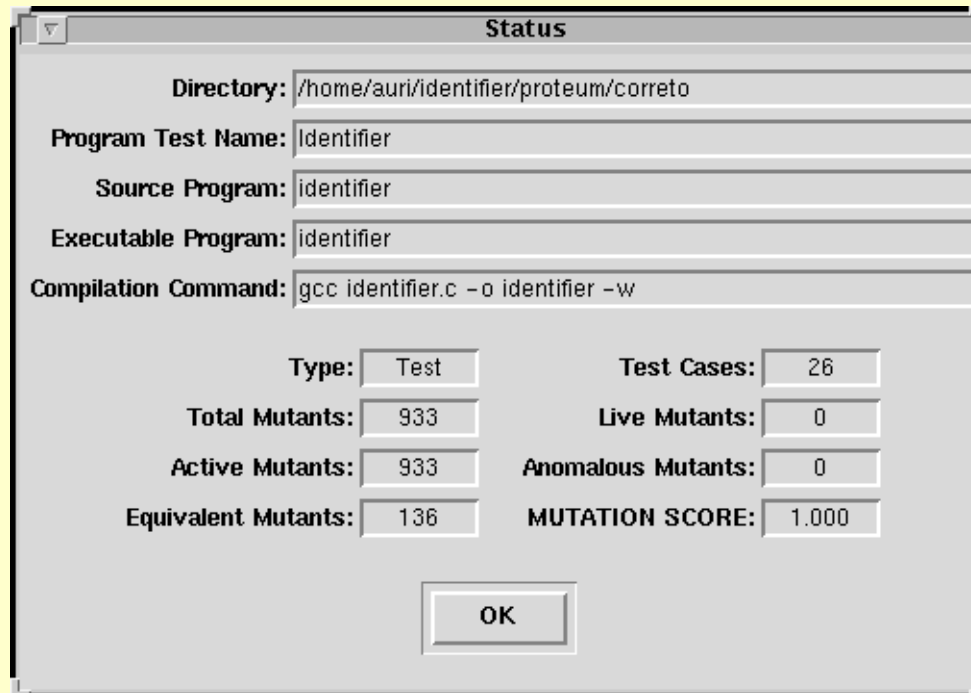
Identifier.c (função *main*)

Versão Corrigida

```
/* 01 */ {
/* 01 */     char  achar;
/* 01 */     int   length, valid_id;
/* 01 */     length = 0;
/* 01 */     printf ("Digite um possível identificador\n");
/* 01 */     printf ("seguido por <ENTER>: ");
/* 01 */     achar = fgetc (stdin);
/* 01 */     valid_id = valid_s(achar);
/* 01 */     if (valid_id)
/* 02 */         length = 1;
/* 03 */     achar = fgetc (stdin);
/* 04 */     while (achar != '\n')
/* 05 */     {
/* 05 */         if (!(valid_f(achar)))
/* 06 */             valid_id = 0;
/* 07 */         length++;
/* 07 */         achar = fgetc (stdin);
/* 07 */     }
/* 08 */     if (valid_id && (length >= 1) && (length <= 6) )
/* 09 */         printf ("Valido\n");
/* 10 */     else
/* 10 */         printf ("Invalido\n");
/* 11 */ }
```

Análise de Mutantes

- *Status* após T_5 no programa corrigido



The screenshot shows a dialog box titled "Status" with the following fields and values:

Directory:	/home/auri/identifier/teum/correto		
Program Test Name:	Identifier		
Source Program:	identifier		
Executable Program:	identifier		
Compilation Command:	gcc identifier.c -o identifier -w		
Type:	Test	Test Cases:	26
Total Mutants:	933	Live Mutants:	0
Active Mutants:	933	Anomalous Mutants:	0
Equivalent Mutants:	136	MUTATION SCORE:	1.000

OK

- $T_5 = T_4 \cup \{(ABCDEF, \text{Válido})\}$

Conclusões

- A atividade de teste é fundamental no processo de desenvolvimento de software
 - Qualidade do produto
- Alto custo da atividade de teste
- Desenvolvimento e aplicação de técnicas e critérios de teste
- Desenvolvimento e utilização de ferramentas de teste
- Não existe um algoritmo de teste de propósito geral para provar a corretitude de um programa