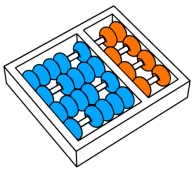
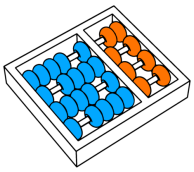


Gizelle Sandrini Lemos

"Alinhamento de Sequências na Avaliação de Resultados de Teste de Robustez"

CAMPINAS
2012



Universidade Estadual de Campinas
Instituto de Computação

Gizelle Sandrini Lemos

"Alinhamento de Sequências na Avaliação de Resultados de Teste de Robustez"

Orientador(a): **Profa. Dra. Eliane Martins**

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.

Este exemplar corresponde à versão final da Tese defendida por Gizelle Sandrini Lemos, sob orientação de Profa. Dra. Eliane Martins.

Assinatura do Orientador(a)

CAMPINAS
2012

FICHA CATALOGRÁFICA ELABORADA POR
MARIA FABIANA BEZERRA MULLER - CRB8/6162
BIBLIOTECA DO INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
COMPUTAÇÃO CIENTÍFICA - UNICAMP

Lemos, Gizelle Sandrini, 1975-
L544a Alinhamento de sequências na avaliação de resultados de teste de robustez / Gizelle Sandrini Lemos. – Campinas, SP : [s.n.], 2012.

Orientador: Eliane Martins.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

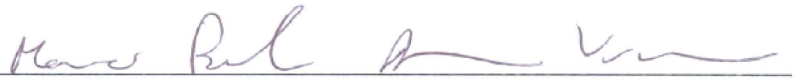
1. software - testes. 2. engenharia de software. I. Martins, Eliane,1955-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em inglês: Sequence alignment algorithms applied in the evaluation of robustness testing results
Palavras-chave em inglês:
Computer software - Testing
Software engineering
Área de concentração: Ciência da Computação
Titulação: Doutora em Ciência da Computação
Banca examinadora:
Eliane Martins [Orientador]
Marco Paulo Amorin Vieira
Márcio Eduardo Delamaro
Guilherme Pimentel Telles
Arnaldo Vieira Moura
Data de defesa: 11-12-2012
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

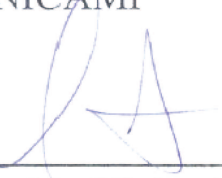
Tese Defendida e Aprovada em 11 de Dezembro de 2012, pela Banca examinadora composta pelos Professores Doutores:



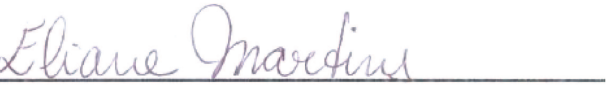
Prof. Dr. Marco Paulo Amorim Vieira
DEI / UC

Prof. Dr. Márcio Eduardo Delamaro
ICMC / USP

Prof. Dr. Guilherme Pimentel Telles
IC / UNICAMP



Prof. Dr. Arnaldo Vieira Moura
IC / UNICAMP



Prof^a. Dr^a. Eliane Martins
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Alinhamento de Sequências na Avaliação de Resultados de Teste de Robustez

Gizelle Sandrini Lemos[1]

11 de Dezembro de 2012

Banca Examinadora:

- Profa. Dra. Eliane Martins (*Orientadora*)
- Prof. Arnaldo Vieira Moura
  IC - UNICAMP
- Prof. Dr. Gulherme Pimentel Telles
  IC - UNICAMP
- Prof. Dr. Márcio Eduardo Delamaro
  ICMC - USP
- Prof. Dr. Marco Paulo Amorim Vieira
  DEI - Universidade de Coimbra
- Prof. Dr. João Meidanis
  IC - UNICAMP (*Suplente*)
- Prof. Dr. João Batista Camargo Júnior
  PCS/Poli - USP (*Suplente*)

[1]Suporte financeiro de: Capes (convênio DS-00014/07-9) 2008–2009, CNPq (processo 142870/2009-9) 2009–2012, Projeto RobustWeb (nro. 623/09) 2009–2011

Abstract

Robustness, which is the ability of a system to work properly in unexpected situations, is an important characteristic, especially for critical systems. A commonly used technique for robustness testing is to inject faults during the system execution and to observe its behavior. A frequent problem during the tests is to determine an oracle, i.e., a mechanism that decides if the system behavior is acceptable or not. Oracles such as the golden run comparison - system execution without injection of faults - consider all different behaviors from the golden run as errors in the system under test (SUT). For example, the activation of exception handlers in the presence of faults could be considered as errors. Safety property searching approach is also used as oracle and it can show the presence of non-robustness in the SUT. If there are events in the SUT execution that are semantically similar to the property they are not taken into account (unless they have been explicitly defined in the property). The objective of this work is to develop specific oracles to evaluate results of robustness testing in order to minimize the deficiences in the current oracles. The main difference between our solutions and the existing approaches is the type of algorithm that we used to compare the sequences. We adopted sequence alignment algorithms commonly applied in Bioinformatics. These algorithms are a kind of inexact matching, allowing some variations between the compared sequences. First approach is based on the tradicional golden run comparison, but applies global sequence alignment of sequences to compare traces collected during fault injection and traces collected without fault injection. The use of this algorithms allows that traces with some differences of the golden run also being classified as robust allowing it use in non-deterministic systems evaluation which is not possible currently. The second approach works with comparison of patterns derived from safety properties and traces collected during robustness testing. However, differently from the first approach, the second one use of local sequence alignment algorithm to search for subsequences. Besides the advantages of the inexact matching, these algorithms use a scoring system based on information obtained from SUT specification to guide the alignment of the sequences. We show the results of the approaches application through case studies.

Resumo

A robustez, que é a capacidade do sistema em funcionar de maneira adequada em situações inesperadas, é uma propriedade cada vez mais importante, em especial para sistemas críticos. Uma técnica bastante empregada para testar a robustez consiste em injetar falhas no sistema e observar seu comportamento. Um problema comum nos testes é a determinação de um oráculo, i.e., um mecanismo que decida se o comportamento do sistema é ou não aceitável. Oráculos como a comparação com padrão-ouro - execução sem injeção de falhas - consideram todo comportamento diferente do padrão como sendo erro no sistema em teste (SUT). Por exemplo, a ativação de rotinas de tratamento de exceção em presença de falhas, podem ser considerados como erros. Também utilizada como oráculo, a busca por propriedades de segurança (safety) pode mostrar a presença de não robustez no SUT. Caso haja no SUT eventos semanticamente similares aos da propriedade estes não são notados (a menos que sejam explicitamente definidos na propriedade). O objetivo deste trabalho é o desenvolvimento de oráculos específicos para teste de robustez visando a diminuição dos problemas existentes nas soluções atualmente empregadas. Desenvolvemos duas abordagens a serem utilizadas como oráculos. O principal diferencial de nossas soluções em relação aos oráculos atuais é a adoção de algoritmos de alinhamento de sequências comumente aplicados em bioinformática. Estes algoritmos trabalham com matching inexato, permitindo algumas variações entre as sequência comparadas. A primeira abordagem criada é baseada na comparação tradicional com o padrão-ouro, porém aplica o alinhamento global de sequências na comparação de traços de execução coletados durante a injeção de falhas e padrões coletados sem a injeção de falhas. Isto permite que traços com pequenas porções diferentes do padrão sejam também classificados como robustos possibilitando inclusive a utilização da abordagem como oráculo em sistemas não deterministas, o que não é possível atualmente. Já, a segunda abordagem busca propriedades de segurança (safety) em traços coletados durante a injeção de falhas por meio do uso do algoritmo de alinhamento local de sequências. Além das vantagens do fato de serem algoritmos de matching inexato, estes algoritmos utilizam um sistema de pontuação que se baseia em informações obtidas na especificação do sistema em teste para guiar o alinhamento das sequências. Mostramos o resultado da aplicação das abordagens em estudos de caso.

Agradecimentos

A Deus em primeiro lugar por me dar saúde, capacidade e força necessárias para o desenvolvimento deste trabalho.

À minha orientadora Eliane Martins que é um exemplo de competência e de dedicação ao trabalho acadêmico, por seu suporte em todos os momentos, por seus conselhos e ideias.

Ao meu marido Mario Prado que, nestes anos em que eu estive imersa neste trabalho, traduziu de maneira perfeita a palavra "companheiro".

Aos meus filhos Marinho e Pedro pela compreensão nas muitas ausências da mamãe durante as brincadeiras e os passeios.

Aos meus pais, Ernani e Cidinha pelo suporte incondicional que eu nunca esquecerei. A ajuda deles foi decisiva na conclusão deste trabalho.

Aos meus irmãos Marcos e Amanda pela valiosa convivência, embora menor do que eu gostaria que fosse.

Aos meus familiares que, mesmo distantes, sempre me apoiaram e torceram pelo meu sucesso.

Aos professores e aos colegas do IC pela ajuda, convivência e dicas sempre bem vindas.

Aos funcionários do IC que sempre se mostraram prontos a nos auxiliar na resolução de dúvidas e problemas.

Aos órgãos de fomento e projetos de pesquisa por meio dos quais obtive auxílio financeiro para a execução deste trabalho.

A todos o meu sincero agradecimento!

Sumário

Lista de Tabelas

Lista de Figuras

Para chegar à realidade, uma ideia começa por se apoderar de espíritos fervorosos e escraviza-os; a partir desse momento, eles pertencem-lhe e não veem diante de si senão o objetivo de atingi-la. Por vezes, esse objetivo parece inatingível: quanto mais nos adiantamos, mais ele nos parece distante. Mas que importa? Os escravos de uma ideia são incapazes de desanimar.

—*Marie Curie*

Capítulo 1

Introdução

Este trabalho está inserido no contexto da avaliação de resultados obtidos durante a execução de testes, mais especificamente na área de testes de robustez. O teste de robustez é uma técnica na qual o comportamento do sistema em teste (SUT - do inglês *System Under Test*) é avaliado na presença de entradas inválidas ou quando inserido em condições hostis [50]. No teste de robustez, a análise de resultados não é trivial pois na grande maioria dos sistemas não há documentação sobre como deveriam se comportar na presença de entradas inválidas.

Os mecanismos comumente utilizados na avaliação dos resultados de teste de robustez empregam algoritmos de matching exato na comparação do comportamento esperado com o comportamento observado durante os testes [83]. Os algoritmos de matching exato realizam apenas análise léxica na busca pelo padrão desejado. Com este tipo de algoritmo diferenças relacionadas ao disparo de mecanismos de tolerância a falhas ou ao indeterminismo do SUT são errôneamente consideradas como problemas da implementação do SUT. Esta característica leva a uma alta taxa de falso-positivos nos resultados avaliados pois apenas o que é exatamente igual ao padrão recebe o veredicto "passou".

Os testes de robustez são de extrema importância, principalmente para sistemas críticos. Assim, a necessidade de tornar a avaliação dos resultados obtidos em testes de robustez mais adequada a este tipo de teste é urgente e este trabalho vem ao encontro desta demanda.

1.1 Objetivo do Trabalho

Este trabalho tem como objetivo contribuir para a melhoria das soluções existentes para avaliação de resultados do teste de robustez. Para isto, desenvolvemos duas novas abordagens que podem ser utilizadas na análise de resultados deste tipo de teste. Nossa pesquisa aplica algoritmos de alinhamento de pares de sequências comumente utilizados na bioin-

formática no contexto do oráculo de teste. Visamos, com isto, obter o mesmo tipo de benefício que a aplicação destes algoritmos apresenta quando utilizados no alinhamento de sequências biológicas.

Os algoritmos de alinhamento de sequências mensuram a similaridade entre cada par de símbolos das sequências comparadas por meio de um sistema de pontuação, que pode incorporar informações sobre o domínio do conhecimento durante a comparação. Estes algoritmos pertencem ao grupo dos algoritmos de matching aproximado, ou seja, permitem algumas diferenças entre a sequência esperada e a sequência observada. O alinhamento entre as sequências é guiado pelo sistema de pontuação, que pode agregar aspectos semânticos à busca [26].

1.2 Terminologia Utilizada

No Brasil, ainda não há consenso sobre alguns termos relativos a testes de software. Descrevemos nesta seção a terminologia adotada nesse trabalho. Os significados associados a cada termo a seguir foram baseados em [50, 7, 64, 88]:

- Falha (do inglês *fault*): passo, processo ou definição de dados incorreto num programa de computador;
- Erro (do inglês *error*): diferença entre o valor computado, observado ou medido e o valor (ou condição) verdadeiro, especificado ou teoricamente correto;
- Defeito (do inglês *failure*): resultado da falha; a inabilidade do sistema ou componentes em desempenhar suas funções requeridas dentro dos requisitos especificados;
- Perigo (do inglês *hazard*): conjunto de condições (estados) que tem risco de conduzir a um incidente, dadas certas condições ambientais. Em teste de robustez, um perigo é denotado por qualquer evento não esperado na especificação nominal do SUT;
- Segurança (do inglês *safety*): definida em termos de perigos ou estados do sistema que, quando combinados com certas condições ambientais, poderiam levar a um incidente;
- Fluxo de controle (do inglês *control flow*): a sequência em que as operações são executadas durante a execução do sistema em teste;
- Fluxo de dados (do inglês *data flow*): a sequência em que transferência, uso e transformação de dados ocorrem durante a execução do sistema em teste;

- Tolerância a falhas (do inglês *fault tolerance*): a habilidade de um sistema ou componente em continuar sua operação normal apesar da presença de falhas de software ou hardware;
- Tolerância a erros (do inglês *error tolerance*): a habilidade de um sistema ou componente em continuar sua operação normal apesar da presença de entradas errôneas;
- Injeção de falhas (do inglês *fault injection*): técnica amplamente utilizada para testes de robustez, que visa verificar como o SUT se comporta na presença de falhas;
- Workload: conjunto de entradas que disparam as atividades do SUT, por exemplo, um conjunto de casos de testes. Neste trabalho, é identificado pelo símbolo $\mathcal{C}$;
- Faultload: conjunto de falhas injetadas no SUT. Identificamos o faultload pelo símbolo $\mathcal{F}$;
- Traço de execução (do inglês *execution trace*): também referenciado como rastro de execução, um traço é uma sequência de eventos executada pelo SUT durante um caso de teste. Um conjunto de traços de execução é identificado neste trabalho pelo símbolo τ;
- Padrão-ouro (do inglês *golden run*): traço de execução do SUT utilizado como modelo de comportamento da execução do sistema para um caso de teste. Identificamos um conjunto de padrões-ouro com o símbolo ρ;
- Log de execução: todos os traços que compõem o conjunto τ ou o conjunto ρ.

No contexto da análise de resultados de teste de robustez, o objetivo é verificar, nos traços de execução coletados do SUT, a existência de erros provenientes da ativação de falhas no código que levam à ocorrência de defeitos.

1.3 Motivação

O mecanismo por meio do qual é realizada a análise de resultados de teste é chamado de oráculo [46]. O problema de decidir se o resultado da execução do teste foi ou não aceitável é conhecido como *Problema do Oráculo* e é um problema indecidível pois não há um oráculo único capaz de obter corretamente o resultado de todo e qualquer teste [66]. Atualmente, são utilizados oráculos específicos para cada problema e muitas vezes pouco adequados ao tipo de teste aplicado [54, 56].

A dificuldade em analisar o resultado dos testes pode ser verificada em teste de robustez, no qual as entradas não pertencem ao conjunto das entradas válidas ao SUT (cujas

saídas são geralmente conhecidas) ou sob condições estressantes. Assim, quando aplicamos na avaliação de resultados de teste de robustez oráculos tradicionalmente usados em testes de conformidade (que analisam o comportamento do sistema em função das entradas válidas) percebemos que os resultados deixam a desejar.

Nossa principal motivação é preencher a lacuna existente na análise de resultados de testes de robustez. Outra limitação dos oráculos atuais é a falta de informações sobre os erros encontrados. Algumas vezes sabe-se, por meio do oráculo, que o traço de execução analisado contém um erro, porém nada mais é informado. A visualização do erro e do contexto em que ocorreu é de extrema importância para o testador. Pretendemos então, com a análise dos traços de execução por meio de algoritmos de alinhamento de sequências, não somente apontar a presença de um erro mas também mostrar as regiões do traço que contêm diferenças em relação ao comportamento desejado, de forma a auxiliar tanto testadores quanto desenvolvedores no diagnóstico de falhas.

1.4 Contribuições deste Trabalho

Podemos citar como contribuições deste trabalho:

- A definição de uma abordagem GCSpec (Golden-run Comparison based on the Specification) que pode ser empregada como oráculo em teste de robustez. A GCSpec estende a comparação com o padrão-ouro substituindo algoritmos de matching exato na comparação por matching inexato realizado com o alinhamento global de sequências;

- A definição de abordagem PSSpec (Property Search based on the Specification) que pode ser empregada também como oráculo em teste de robustez de forma complementar à GCSpec. A PSSpec emprega o algoritmo de alinhamento local de sequências para procurar por um padrão de comportamento no traço, que representa um padrão baseado na quebra de uma propriedade de segurança (*safety*);

- A definição de um método que pode ser aplicado no desenvolvimento do sistema de pontuação para o algoritmo de alinhamento de sequências. O sistema de pontuação é também, além das duas sequências a serem comparadas, uma entrada para o algoritmo de alinhamento e os valores deste sistema de pontuação expressam o relacionamento entre cada par de símbolos que compõem as sequências. A qualidade do sistema de pontuação interfere no resultado final do alinhamento. Por este motivo criamos um método adequado à definição do sistema de pontuação de sequências que representam o comportamento de sistemas em teste;

- A classificação qualitativa dos traços de execução analisados. Cada traço é classificado conforme uma métrica proposta neste trabalho, obtida a partir da pontuação do alinhamento;
- Uma ferramenta que implementa as abordagens GCSpec e PSSpec. Com o uso da ferramenta é possível obter o alinhamento entre um traço de execução e um padrão de comportamento que pode ser um traço de execução coletado sem a injeção de falhas ou uma propriedade de segurança (*safety*). Para ambas as abordagens a ferramenta apresenta a visualização do alinhamento entre as duas sequências.

1.5 Organização do Texto

O Capítulo 2 introduz os conceitos necessários para o entendimento do teste de robustez, seus resultados e a análise da forma como é realizada atualmente. O Capítulo 3 apresenta o alinhamento de sequências e os principais algoritmos de programação dinâmica utilizados para este fim. No Capítulo 4 definimos um sistema de pontuação adequado ao alinhamento de sequências compostas por eventos de sistemas de software. No Capítulo 5 descrevemos a primeira abordagem desenvolvida, que estende os conceitos da comparação com padrão-ouro por meio do uso de algoritmos de alinhamento global (ou semi-global) de sequências. No Capítulo 6 descrevemos a segunda abordagem desenvolvida que, utiizando o algoritmo de alinhamento local, procura por propriedades de segurança (*safety*) nos traços de execução coletados durante o teste de robustez. O Capítulo 7 apresenta a ferramenta construída com o objetivo de implementar as abordagens descritas neste trabalho. Por fim, no Capítulo 8 tecemos as conclusões desta pesquisa e descrevemos as perspectivas quanto a trabalhos futuros.

Capítulo 2

Teste de Robustez

Neste capítulo encontra-se a fundamentação teórica sobre teste de robustez e os métodos de análise de resultados atualmente empregados, seus problemas e suas limitações.

A Seção 2.1 contém a introdução sobre a verificação da robustez de um software. Na Seção 2.2 detalhamos o teste de robustez. Em seguida, na Seção 2.3 apresentamos os oráculos de teste, em especial os oráculos atualmente utilizados na avaliação de resultados de testes de robustez. Por fim, temos as considerações finais do capítulo na Seção 2.4.

2.1 Introdução

Atualmente, tão importante quanto determinar se um sistema se comporta de acordo com a especificação (objetivo do teste de conformidade) é definir se o sistema se comporta de maneira aceitável fora do contexto especificado, ou seja, na presença de entradas inválidas ou sob condições estressantes. Assim, a realização de testes de robustez, que visa verificar o comportamento do sistema fora do contexto descrito na especificação, vem se tornando um passo comum para a validação de sistemas.

Uma das principais técnicas utilizadas na execução de testes de robustez é a injeção de falhas [103, 73]. Nesta técnica, falhas são deliberadamente injetadas no sistema em teste para verificar como este se comporta [8]. As três principais categorias de injeção de falhas são [68, 58]:

- Injeção de falhas via simulação (*Simulation Fault Injection*): permite, na fase de projeto, testar os mecanismos de tolerância a falhas;
- Injeção de falhas via hardware (*Hardware Fault Injection*): utiliza um hardware adicional para injetar falhas de hardware no sistema em teste;
- Injeção de falhas via software (*Software Fault Injection*):

- Interface: em tempo de execução, entradas inválidas e não especificadas são inseridas no SUT;
- Código-fonte: pode ser estática ou dinâmica. A injeção de falhas estática não necessita de software auxiliar para ser executada pois a falha é injetada no código fonte do SUT em tempo de compilação e é ativada quando a parte do código em que a falha foi injetada é executada. Já, a injeção de falhas dinâmica necessita de software auxiliar para ser aplicada (injetor de falhas e monitor) pois as falhas são injetadas em tempo de execução.

Os testes de robustez realizados nos estudos de caso dos quais coletamos os traços utilizados nesta tese aplicaram a técnica de injeção de falhas por software (estática e dinâmica). Maiores detalhes podem ser encontrados nos Capítulos 5 e 6.

Um sistema é considerado "robusto" quando se comporta de maneira aceitável na presença de entradas inválidas ou sob condições estressantes do ambiente [89]. Porém, a interpretação dos resultados obtidos no teste de robustez não é precisa como no teste de conformidade onde as saídas esperadas para o conjunto de entradas fornecidas são previamente conhecidas.

No teste de conformidade, se os resultados são diferentes do especificado, podemos afirmar que o teste revelou a presença de uma falha no SUT. Porém, quando na presença de entradas inválidas, o comportamento esperado do sistema geralmente não é especificado. Desta forma, como verificar se o resultado obtido no teste de robustez é aceitável? Os oráculos que vêm sendo empregados para avaliar resultados de teste de robustez não são indicados para este tipo de teste pois, no contexto do teste de robustez, nem todo resultado diferente do esperado é errôneo.

2.2 Teste de Robustez

No teste de robustez, diferentemente do teste de conformidade, não há preocupação com a corretude do SUT do ponto de vista funcional, mas sim do ponto de vista da robustez do SUT em relação à resposta para entradas inválidas, perigos (do inglês *hazards*) ou condições ambientais estressantes [87, 50, 38].

O teste de robustez pode ser realizado em várias fases do ciclo de vida do sistema, desde a fase de desenvolvimento até as fases de validação e verificação [72]. Testes de robustez também são indicados para a avaliação de componentes de software utilizados em situações diferentes daquelas para as quais eles foram originalmente projetados, como forma de verificar se os mesmos se comportam de forma adequada em outros contextos [60].

A injeção de falhas é a principal técnica utilizada para a realização de testes de robustez. Na injeção de falhas, o domínio de entrada para o teste é dividido em *workload*,

conjunto de casos de teste com entradas especificadas $\mathcal{C} = c_1, c_2, \ldots, c_r$, sendo $r \in N^*$ o número de casos de teste e *faultload*, conjunto de falhas $\mathcal{F} = f_1, f_2, \ldots, f_s$, sendo $s \in N^*$ o número de falhas aplicadas ao SUT durante o teste.

Cada uma das s falhas do conjunto $\mathcal{F}$ é exercitada para cada um dos r casos de teste do workload ($\mathcal{C}$). Assim, ao final do processo de teste temos, para cada SUT, um conjunto de traços de execução (τ) formando um log de teste de tamanho $|r * s|$.

As falhas injetadas na interface do SUT pertencem ao conjunto de valores de entrada que extrapolam os limites do que é considerado válido para o domínio de entrada, conforme ilustrado pela Figura 2.1 [35, 54]. No trabalho de Vieira e colegas [97], os autores combinam entradas válidas e inválidas em sistemas web com o intuito de verificar se tais entradas são capazes de disparar erros devido a falhas internas do SUT.

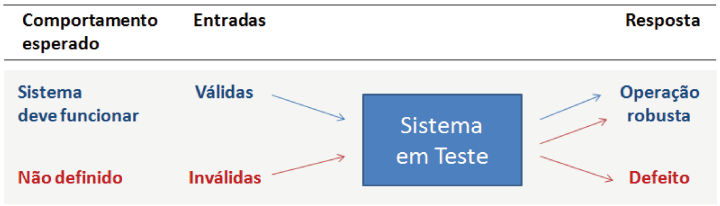


Figura 2.1: Abordagem utilizada em testes de robustez (adaptado de [54])

Segundo o relatório do Software Engineering Institute [23], a aplicação de testes de robustez durante os testes de unidades e de integração pode auxiliar tanto na descoberta de problemas de codificação quanto na melhoria do modelo de tratamento de erros.

A técnica de mutação de partes do código executado pelo SUT é um tipo de injeção de falhas estática usada no teste de robustez com o intuito de simular falhas de programação [72]. Além da mutação, podem ainda ser injetados erros, emulando as consequências das falhas de software manipulando o estado do sistema em teste ou corrompendo os valores de parâmetros do sistema [30, 56, 103].

2.3 Oráculos de Teste

A última etapa do ciclo de vida de teste é a análise dos resultados [29]. Para realização desta análise é necessário o uso de um oráculo de teste. O termo oráculo foi introduzido por Howden em 1978 [46]. Segundo ele, um oráculo pode ser utilizado para checar a corretude dos valores de saída para um determinado conjunto de entradas. De acordo com Binder, o oráculo é uma referência de resultados para os testes [14]. Porém, infelizmente, não há um oráculo único capaz de verificar se a saída está correta para toda entrada exercitada no SUT. Este problema é chamado de *Problema do Oráculo* [66] e ainda encontra-se em aberto.

Segundo Richardson et al. [86], o oráculo pode ser decomposto em duas partes:

1. A *informação do oráculo* que especifica qual é o comportamento correto para o SUT;
2. O *procedimento do oráculo* que compara os resultados da execução do teste com a informação do oráculo.

Os oráculos podem também ser classificados como manuais ou automatizados [105]. O tipo mais simples de oráculo é o oráculo manual, no qual o testador faz o papel de procedimento do oráculo comparando os resultados da execução dos testes com a especificação do sistema. Este tipo de oráculo apresenta limitações como lentidão e propensão a erros. Quanto mais complexo é o SUT mais difícil é a ánálise com oráculo manual [9].

Há vários tipos de oráculos automatizados. Um dos tipos mais utilizados é a comparação de cada traço de execução coletado durante o teste com o padrão-ouro correspondente a cada caso de teste [83]. O padrão-ouro é um traço de execução utilizado como padrão de comportamento do SUT para um determinado caso de teste. Em testes de conformidade, o padrão-ouro é uma outra versão do sistema ou uma implementação legada exercitada com o mesmo workload. Para o teste de robustez com injeção de falhas, o padrão-ouro é a execução do SUT sem a injeção de falhas (o SUT é exercitado apenas com o workload [45]).

Porém, há alguns problemas com este tipo de oráculo. Ele assume que o comportamento do SUT é reprodutível, ou seja, a cada execução do mesmo workload (com ou sem injeção de falhas), o comportamento do SUT não deveria se alterar. Porém, o comportamento do SUT com e sem injeção de falhas, na maioria das vezes, não é exatamente igual pois um sistema robusto dispara mecanismos de tratamento de falhas durante a injeção de falhas com o objetivo de tratá-las.

O disparo de mecanismos de tolerência a falhas ou mesmo indeterminismos do SUT tornam as duas execuções (com e sem injeção de falhas) diferentes mesmo para um mesmo SUT. Neste caso, a diferença entre as duas execuções não indica que o sistema não é robusto. O fato deste tipo de oráculo considerar com o veredicto "falhou" todo traço diferente do padrão-ouro leva a uma alta taxa de falso-positivos (traços classificados como errôneos mas que na verdade não apresentam erros).

Outro tipo de oráculo, derivado da comparação com o padrão-ouro, é a comparação dos resultados do teste com *execuções de referência* [58]. A execução de referência é um traço de execução utilizado como modelo de comportamento para o SUT. A diferença em relação ao padrão-ouro está no fato de que este tipo de modelo reúne tanto o fluxo de controle quanto o fluxo de dados e a execução de referência aceita, para cada parâmetro, um intervalo de valores ao invés de um único valor.

Entretanto, os valores válidos de cada intervalo são obtidos a partir de execuções-modelo, o que significa que, se o número de execuções for insuficiente, os intervalos gerados não representarão com fidelidade os valores possíveis de cada parâmetro. Traços do SUT

com valores de parâmetros fora dos intervalos considerados válidos são classificadas como errôneos. Assim como na comparação tradicional com o padrão-ouro, este oráculo não permite desvio na sequência de eventos esperados.

Outra abordagem, desenvolvida por Andrews, utiliza uma linguagem para representar a descrição de máquinas de estados utilizadas com o intuito de verificar a corretude de logs de execução [4]. As máquinas de estados são compostas por eventos (e seus parâmetros) obtidos da especificação. O oráculo baseado em máquinas de estados precisa identificar se há um ponto do log analisado que corresponde ao evento inicial da máquina para então fazer a correspondência entre os demais eventos do log e os estados subsequentes da máquina. Para ser usado em testes de robustez, este tipo de oráculo precisa conseguir descrever estados e transições não especificados, o que não é tarefa trivial [88].

Tanto a comparação do traço de execução do SUT com o padrão-ouro (ou com a execução de referência) como o oráculo por máquinas de estado são usados após a execução dos testes, ou seja, tratam-se de oráculos off-line. Há oráculos on-line como a verificação de assertivas em tempo de execução (do inglês *runtime assertions*). Esse oráculo consiste da inserção de assertivas no código-fonte do SUT para que sejam verificadas enquanto os testes são realizados [9, 6].

Um dos problemas do oráculo por assertivas é que algumas das assertivas são propriedades que devem ser válidas em pontos específicos da execução do SUT enquanto outras devem ser válidas em todas as execuções, o que torna difícil a verificação. Outro problema é que nem todo caso de teste presente no workload pode resultar em erro do SUT enquanto outros podem resultar em erros em trechos posteriores da execução, não verificados por assertivas.

A busca por propriedades em traços de execução é um tipo de oráculo que pode ser empregado na análise de resultados de teste de robustez. Apesar da busca por propriedades ser aparentemente similar à verificação de assertivas, as propriedades apresentam algumas diferenças marcantes em relação às assertivas:

- As propriedades são procuradas após o encerramento dos testes e não durante a execução do SUT;

- As propriedades descrevem comportamentos (corretos ou não) que devem ser satisfeitos (ou não) pelo SUT. Tais comportamentos podem ser obtidos, por exemplo, a partir da especificação do SUT ou por meio de análises de segurança (*safety*) enquanto que as assertivas descrevem estados do programa num determinado momento da execução;

- As propriedades não são inseridas no código do SUT, como ocorre com as assertivas. Elas são comparadas com os traços de execução coletados ao longo do teste.

Cavalli et al. descrevem uma abordagem de avaliação de resultados de teste de robustez por meio da busca de propriedades. Tais propriedades foram nomeadas pelos autores como "invariantes" pelo fato de que devem ser válidas durante toda a execução do SUT [19]. Na tese de Fayçal Bessayah, o autor desenvolveu uma abordagem para avaliação de teste de robustez por injeção de falhas por meio da busca de cenários de robustez. Segundo o autor, cenários de robsutez podem ser expressos em propriedades de segurança (*safety*), que definem como um sistema robusto deve evitar um cenário perigoso e propriedades de liveness que especificam como o sistema deve reagir a cenários estressantes [12]. Ambos os trabalhos citados utilizaram algoritmos de matching exato durante a busca das propriedades.

No trabalho de David Lo e colegas, os autores utilizaram técnicas baseadas em mineração de dados (*data mining*) para a avaliação de traços de execução na detecção de defeitos. De uma maneira geral, a técnica utilizada requer que o classificador, que emite o veredicto para os traços como sendo ou não anômalos, seja treinado. Os dados de treinamento formam um conjunto que habilita o classificador a julgar os demais traços fornecidos. Outras abordagens similares trabalham verificando a frequência de ocorrência dos eventos, assumindo que o comportamento "normal" é mais frequente que o comportamento anômalo. Seja com dados de treinamento ou por meio da frequência dos eventos, a calibragem do algoritmo para a diferenciação entre comportamentos normais e errôneos não é trivial. Outro problema com os algoritmos de mineração de dados é que eles são sensíveis a ruídos, ou seja, eventos não diretamente relacionados aos comportamentos mais frequentes do sistema mas que aparecem nos traços analisados e que podem induzir o classificador ao erro [65].

Alguns trabalhos não utilizam oráculos para verificar todos os comportamentos do sistema em relação às entradas fornecidas durante os testes [35]. No caso de testes de robustez, há autores que se baseiam na afirmação de que só há sinal de não robustez quando ocorre um evento de erro como colapso (*crash*), bloqueio (*hang*) ou aborto (*abort*). Baseados neste tipo de premissa, ao invés de utilizarem oráculos, classificam os traços utilizando escalas como a CRASH [54]. A escala CRASH classifica os defeitos de robustez em:

- Catastrófico (*Catastrophic*): ocorre quando o sistema operacional é corrompido;
- Reinicialização (*Restart*): o sistema fica bloqueado e precisa ser abortado;
- Aborto (*Abort*): o sistema termina de forma anormal. Aborta por si só;
- Silêncio (*Silent*): houve algum problema mas nada é reportado pelo sistema;
- Obstrução (*Hindering*): o código de erro reportado não condiz com o erro ocorrido.

Porém, durante o teste de robustez, o sistema pode falhar sem que ocorra nenhum dos problemas acima citados. Como exemplo de uma falha sem a ocorrência de um evento de erro podemos citar um sistema que controla um grupo de elevadores. Um defeito de robustez ocorre se um dos elevadores abrir a porta antes de chegar ao andar solicitado pelo usuário. Neste caso, o traço não contém nenhum evento de erro porém podemos afirmar, a partir da ocorrência citada, que o sistema não é robusto.

2.4 Considerações Finais

A qualidade do oráculo é de extrema importância para o processo de teste. Quando lidamos com teste de robustez, além deste aspecto, precisamos levar em conta também a questão da adequação do oráculo a este tipo de teste. Um oráculo inadequado pode levar o testador a acreditar que o SUT é robusto quando, na verdade, não é. Esta importante técnica de teste necessita de um oráculo adequado à verificação de comportamentos resultantes da injeção de falhas.

Algumas características podem ser utilizadas para classificar um oráculo quanto à adequação na avaliação de resultados de teste de robustez. A sensibilidade (número de decisões positivas em relação a todos os casos verdadeiramente positivos) e a especificidade (número de decisões negativas em relação a todos os casos verdadeiramente negativos) são exemplos que, juntos, podem auxiliar na avaliação da qualidade de um oráculo. Nos Capítulos 5 e 6 apresentamos os resultados dos oráculos propostos considerando estas características.

Capítulo 3

Alinhamento de Sequências

O alinhamento de pares de sequências é uma técnica usada para comparar duas sequências sobrepondo uma à outra de forma a localizar semelhanças e diferenças entre elas. Há, geralmente, várias maneiras de alinhar um mesmo par de sequências. Dentre os alinhamentos possíveis, melhor será aquele capaz de exibir a maior quantidade de posições coincidentes entre as duas sequências.

A Seção 3.1 descreve as categorias de alinhamentos entre pares de sequências. A Seção 3.2 define a terminologia e os conceitos básicos necessários para a descrição dos algoritmos de alinhamento de pares de sequências. A Seção 3.3 descreve o algoritmo de alinhamento global. A Seção 3.4 trata do algoritmo semi-global, variação do algoritmo de alinhamento global que não penaliza os espaços nas extremidades das sequências. O algoritmo de alinhamento local é descrito na Seção 3.5. Na Seção 3.6 mostramos a recuperação de mais de um alinhamento a partir de um par de sequências. A Seção 3.7 contém a discussão sobre algumas variações no sistema que pontua as escolhas feitas pelo algoritmo de alinhamento. Finalmente, na Seção 3.8 temos as considerações finais.

3.1 Introdução

O procedimento de alinhar sequências não tem como objetivo apenas encontrar sequências totalmente idênticas. O principal objetivo do alinhamento é encontrar trechos não idênticos porém similares de acordo com algum critério pré-estabelecido. Por este motivo, algoritmos de matching exato não são adequados pois são capazes apenas de classificar os símbolos de duas sequências como idênticos ou diferentes, ou seja, não são capazes de mensurar o grau de similaridade entre eles. O alinhamento de símbolos diferentes porém similares pode ser obtido através de algoritmos de matching inexato.

Os algoritmos de matching inexato, também chamados de algoritmos de matching aproximado, comparam duas sequências permitindo algumas diferenças entre elas [39].

Na bioinformática, algoritmos de matching inexato são utilizados para o alinhamento de sequências de DNA, RNA e proteínas.

Os principais algoritmos de matching inexato para o alinhamento de um dado par de sequências (x, y) são baseados em programação dinâmica ou em heurísticas [39]. Os algoritmos heurísticos são mais velozes, mas sua aplicação não garante a localização do alinhamento ótimo entre as sequências envolvidas. Já, os algoritmos baseados na técnica de programação dinâmica [24] são mais lentos porém garantem, para um dado sistema de pontuação, a localização do alinhamento ótimo entre as sequências [75]. A Tabela 3.1 contém as complexidades de alguns algoritmos de cada classe.

Tabela 3.1: Complexidade de alguns algoritmos de alinhamento de pares de sequências

Algoritmo	Tipo	Complexidade
Needleman and Wunch [78]	Programação dinâmica	$O(mn)$
Smith and Waterman [92]	Programação dinâmica	$O(mn)$
BLAST [3]	Heurístico	$O(mn/20^w)$
FASTA [82]	Heurístico	$O(20^w)$
m - Tamanho da primeira sequência analisada (em número de simbolos) n - Tamanho da segunda sequência analisada (em número de simbolos) w - Tamanho da subsequência de aminoácidos		

O algoritmo FASTA [82], para alinhamento local, é o mais rápido entre os algoritmos citados. Sua complexidade, assim como do algoritmo BLAST [3], foi medida em relação ao alinhamento de sequências de aminoácidos e, por este motivo, se associa aos 20 tipos de aminoácidos existentes na natureza. Maiores detalhes sobre estes algoritmos e outros algoritmos heurísticos podem ser obtidos em [91, 75, 21].

Como o foco desta pesquisa é a avaliação de resultados de teste de robustez, o que requer um alto grau de acurácia, decidimos nos concentrar apenas no estudo dos algoritmos de alinhamento de sequências que utilizam programação dinâmica.

O alinhamento de pares de sequências pode ser classificado de acordo com as seguintes categorias [91]:

- Global: útil quando as sequências apresentam quase o mesmo tamanho. Neste método as sequências são esticadas (com a inserção de espaços quando necessário) para que ambas alcancem o mesmo tamanho;
- Semi-global: variação do alinhamento global em que todos os espaços inseridos antes do início ou após o final das sequências não afetam a pontuação do alinhamento;
- Local: útil quando as sequências apresentam tamanhos muito diferentes. O foco desta categoria de alinhamento é encontrar regiões nas duas sequências que possuam alta similaridade.

Na Figura 3.1, as sequências GTGTATACCAGATG e GTACCCAG foram submeticas às três categorias de alinhamento. "|" representa o alinhamento de dois símbolos idênticos, "⋆" representa o alinhamento entre dois símbolos diferentes e "−" representa a inserção de um espaço numa das sequências.

Global	GTGTATACCAGATG || | *||| | GT--A-CCCA---G
Semi-Global	GTGTATACCAGATG || | *||| GT--A-CCCAG---
Local	GTGTATACCAGATG ||| *||| --GTA-CCCAG---

Figura 3.1: Exemplos de alinhamentos de acordo com as diferentes categorias

3.2 Terminologia e Conceitos Básicos

Esta seção contém a descrição de termos particulares à teoria que envolve o alinhamento de sequências. Os termos aqui definidos são aplicados ao longo dos Capítulos 3, 4, 5 e 6.

3.2.1 Terminologia

Os principais termos associados à descrição do alinhamento de um par de sequências são:

- Alfabeto (Σ): conjunto finito e não vazio de símbolos.
- Símbolo (σ): cada elemento $\sigma_i \in \Sigma$ com $1 \leq i \leq t$, sendo t o número de símbolos em Σ.
- Sequência (x): série de símbolos $x = x_1, x_2, \ldots, x_m$ em que cada um dos m símbolos de x pertence a Σ. O comprimento da sequência x, ou seja, o número de elementos de x, é denotado por $|x| = m$, onde $m \in \mathbb{N}$. O j-ésimo elemento de x é denotado por $x[j]$, onde $1 \leq j \leq m$.
- Match: correspondência entre dois símbolos $x[i]$ e $y[j]$ no alinhamento de um par de sequências (x, y). Neste caso, $x[i] = y[j]$.
- Mismatch: divergência entre dois símbolos $x[i]$ e $y[j]$ no alinhamento de um par de sequências (x, y). Neste caso, $x[i] \neq y[j]$.

- Similaridade: grau de semelhança entre duas sequências x e y.

3.2.2 Conceitos Básicos

Os algoritmos de alinhamento de sequências são comumente utilizados na bioinformática para o alinhamento de sequências de DNA, RNA e proteínas, de forma a identificar regiões de similaridade que possam ser uma consequência de um relacionamento funcional, estrutural ou evolutivo entre elas [75].

Definição 3.1. O *alinhamento de pares de sequências* ($\alpha(x, y)$) é o resultado da comparação de duas sequências x e y sobre um mesmo alfabeto Σ em busca de uma série de símbolos que têm a mesma ordem nas sequências analisadas.

O alinhamento é capaz de exibir de maneira explícita quais regiões são similares ou diferentes entre o par de sequências (x, y) através das seguintes operações (ilustradas na Figura 3.2):

- Inserção de um ou mais símbolos em y em relação à x;
- Deleção de um ou mais símbolos em y em relação à x;
- Mutação de um símbolo em y em relação a um símbolo em x.

As operações de inserção e deleção de símbolos são chamadas "INDEL" (representando INserção e DELeção) a partir desta seção.

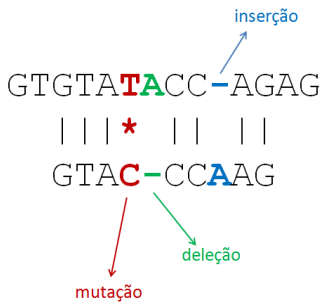


Figura 3.2: Possíveis operações entre pares de símbolos em duas sequências alinhadas

O algoritmo de alinhamento de sequências atribui uma pontuação $\wp_{\alpha\omega}(x, y)$ a cada alinhamento α entre um par de sequências (x, y) conforme um sistema de pontuação ω.

Definição 3.2. Um *sistema de pontuação* (ω) é um conjunto de valores que quantifica cada possível combinação entre pares (s_1, s_2) de símbolos $\in \overline{\Sigma} = \Sigma \cup \{-\}$. Assim:

$$\omega : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{N}$$

O sistema de pontuação é uma entrada importante para o alinhamento de sequências, pois permite pontuar de forma individual cada combinação de pares de símbolos valorizando ou desvalorizando determinados alinhamentos. Assim, a pontuação do alinhamento entre duas sequências x e y é o resultado da soma das pontuações obtidas com a ocorrência de matches, mismatches, inserções e deleções de símbolos.

Todo sistema de pontuação (ω) é composto de duas partes:

1. Matriz de pontuação ($\mathcal{S}$): contém pontuações para todos os pares de símbolos de Σ, recompensando a ocorrência de matches e mismatches;

2. Função de penalidades para ocorrência de espaços (g): penalidades atribuídas às operações de indel no alinhamento. Descrevemos as formas mais utilizadas de penalização de espaços na Seção 3.7.

Dentre os alinhamentos obtidos pelos algoritmos durante o processo de comparação de duas sequências x e y, o alinhamento que melhor representa a similaridade entre elas é dito "ótimo".

Definição 3.3. O *alinhamento ótimo* ($\alpha_{ot(\omega)}$) entre duas sequências x e y é aquele que maximiza a pontuação do alinhamento, ou seja, apresenta pontuação máxima dentre todos os alinhamento produzidos pelo algoritmo para um dado par de sequências considerando um sistema de pontuação ω previamente definido.

$$\alpha_{ot(\omega)} = max(\wp_{\alpha\omega})$$

Pode haver mais de um alinhamento ótimo entre duas sequências. Neste caso, o algoritmo seleciona apenas um deles e descarta os demais. Há algorimos que trabalham selecionando outros alinhamentos a partir de uma mesma matriz de similaridades. Maiores detalhes podem ser obtidos na Seção 3.6.

3.3 Alinhamento Global de Sequências

Um dos algoritmos clássicos de alinhamento entre duas sequências x e y é o algoritmo para alinhamento global desenvolvido por Needleman e Wunsch [78]. Ele alinha a totalidade dos símbolos das duas sequências e, ao final do processo, temos [91]:

$$\alpha_{global} = (x', y') \tag{3.1}$$

em que:

- Os símbolos de x' e $y' \in \Sigma \cup \{-\}$,

- $|x'| = |y'|$,
- $x' = x$ e $y' = y$ quando todos os espaços são removidos,
- $\nexists i, 1 \leq i \leq |x'|$ tal que $x'[i] = y'[i] = -$.

O algoritmo de alinhamento global obtém o alinhamento ótimo em dois passos:

1. Construção da matriz de similaridades (M_s): a matriz, com dimensões $m+1 \times n+1$ onde m e n são, respectivamente, os tamanhos das sequências $x = \{x_1 x_2 ... x_m\}$ e $y = \{y_1 y_2 ... y_n\}$ envolvidas no alinhamento armazena, em cada uma das células $M_s[i, j]$, $0 \leq i \leq m$ e $0 \leq j \leq n$, o alinhamento ótimo entre as subsequências $x_1 ... x_i$ e $y_1 ... y_j$.

2. Recuperação do alinhamento ótimo (do inglês *backtracking*): Este passo do processo, no alinhamento global, é iniciando na célula $M_s[m, n]$ e termina na célula $M_s[0, 0]$ de forma a obter, ao final do passo, um alinhamento entre as duas sequências que contenha a maior pontuação possível dentre os possíveis alinhamentos entre elas.

O preenchimento da matriz M_s, na construção do alinhamento, é realizado conforme as Equações 3.2, 3.3 e 3.4:

$$M_s[i, 0] = i * g \tag{3.2}$$

$$M_s[0, j] = j * g \tag{3.3}$$

$$M_s[i, j] = max \begin{cases} M_s[i, j-1] + g \\ M_s[i-1, j-1] + S[i, j] \\ M_s[i-1, j] + g \end{cases} \tag{3.4}$$

onde S é a matriz de pontuação e g é a função de penalização para a ocorrência de espaços no alinhamento.

O sistema de pontuação associa uma pontuação alta a cada "bom" alinhamento, ou seja, quando há alta similaridade entre os símbolos do par alinhado. Porém, pares de símbolos que possuem baixa similaridade recebem pouca ou nenhuma recompensa quando alinhados.

A Figura 3.3 - item (a) mostra o alinhamento ótimo entre duas sequências $x = GTGTATACCAGATG$ e $y = GTACCCAG$ para o sistema de pontuação mostrado no item (b) da mesma figura. Neste exemplo, a penalidade adotada para a ocorrência de espaços foi $g = -1$. A matriz de similaridades M_s e o alinhamento ótimo recuperado durante o backtracking são mostrados, respectivamente, nos itens (a) e (b) da Figura 3.4.

```
G T G T A T A C C A G A T G
|   |     |   *   | | |       |
G T - - A - C C C A - - - G
+3 +3 -1 -1 +3 -1 +1 +3 +3 +3 -1 -1 -1 +3 = 16
```

(a)

$S =$

	T	G	A	C
T	3	1	0	1
G	1	3	2	0
A	0	2	3	1
C	1	0	1	3

$g = -1$

(b)

Figura 3.3: (a) Alinhamento global ótimo entre as sequências x e y. (b) Matriz de pontuação e penalidade para os espaços utilizados no alinhamento

	–	G	T	A	C	C	C	A	G
–	0	-1	-2	-3	-4	-5	-6	-7	-8
G	-1	3	2	1	0	-1	-2	-3	-4
T	-2	2	6	5	4	3	2	1	0
G	-3	1	5	8	7	6	5	4	4
T	-4	0	4	7	9	8	7	6	5
A	-5	-1	3	7	8	10	9	10	9
T	-6	-2	2	6	8	9	11	10	11
A	-7	-3	1	5	7	9	10	14	13
C	-8	-4	0	4	8	10	12	13	14
C	-9	-5	-1	3	7	11	13	13	13
A	-10	-6	-2	2	6	10	12	16	15
G	-11	-7	-3	1	5	9	11	15	19
A	-12	-8	-4	0	4	8	10	14	18
T	-13	-9	-5	-1	3	7	9	13	17
G	-14	-10	-6	-2	2	6	8	12	16

(a)

	–	G	T	A	C	C	C	A	G
–	0	-1	-2	-3	-4	-5	-6	-7	-8
G	-1	3	2	1	0	-1	-2	-3	-4
T	-2	2	6	5	4	3	2	1	0
G	-3	1	5	8	7	6	5	4	4
T	-4	0	4	7	9	8	7	6	5
A	-5	-1	3	7	8	10	9	10	9
T	-6	-2	2	6	8	9	11	10	11
A	-7	-3	1	5	7	9	10	14	13
C	-8	-4	0	4	8	10	12	13	14
C	-9	-5	-1	3	7	11	13	13	13
A	-10	-6	-2	2	6	10	12	16	15
G	-11	-7	-3	1	5	9	11	15	19
A	-12	-8	-4	0	4	8	10	14	18
T	-13	-9	-5	-1	3	7	9	13	17
G	-14	-10	-6	-2	2	6	8	12	16

(b)

Figura 3.4: (a) Matriz de similaridade (b) Recuperação do alinhamento global ótimo

3.4 Alinhamento Semi-Global de Sequências

O alinhamento semi-global entre duas sequências x e y é uma variação do alinhamento global que não penaliza operações de indel nas extremidades de x ou y [91]. Isto é feito através de algumas modificações no algoritmo de alinhamento global.

Conforme a modificação adotada, são ignorados os espaços em determinada região das sequências. Sendo, respectivamente, x e y a primeira e a segunda sequências no alinhamento, temos as modificações necessárias no algoritmo descritas na Tabela 3.2:

Podemos implementar apenas uma das restrições apresentadas na Tabela 3.2 ou incorporar todas elas ao algoritmo para desconsiderarmos os prefixos e sufixos de ambas as sequências.

Tabela 3.2: Possíveis modificações no algoritmo de alinhamento semi-global [91]

Localização dos espaços	Modificação necessária ma matriz de similaridade
No início de x	Inicializar a primeira coluna com zeros
No início de y	Inicializar a primeira linha com zeros
No final de x	Obter a pontuação máxima na última linha de M_s
No final de y	Obter a pontuação máxima na última coluna de M_s

3.5 Alinhamento Local de Sequências

O algoritmo proposto por Smith e Waterman [92] realiza o alinhamento local entre duas sequências x e y. Assim como os algoritmos global e semi-global, este algoritmo constrói uma matriz de similaridades M_s com dimensões $m+1 \times n+1$ onde $m = |x|$ e $n = |y|$. Porém, para o alinhamento local, o preenchimento de M_s se dá conforme as equações 3.5, 3.6 e 3.7.

$$M_s[i,0] = 0 \tag{3.5}$$

$$M_s[0,j] = 0 \tag{3.6}$$

$$M_s[i,j] = max \begin{cases} M_s[i,j-1] + g \\ M_s[i-1,j-1] + S[i,j] \\ M_s[i-1,j] + g \\ 0 \end{cases} \tag{3.7}$$

Como podemos notar na Equação 3.7, nenhuma célula de M_s recebe valores negativos. Com essa estratégia o algoritmo valoriza o alinhamento de trechos de alta similaridade entre as sequências.

A etapa de recuperação do alinhamento local ótimo inicia a busca na célula de M_s que contém o maior valor dentre todas as células da matriz e termina ao encontrar uma célula que contenha o valor 0.

```
G T G T A T A C C A G A T G
    | | |   * | | | | |
- - G T A - C C C A G - - -
  +3 +3 +3 -1 +1 +3 +3 +3 +3        = 21
```

Figura 3.5: Alinhamento local ótimo

A Figura 3.6, partes (a) e (b) mostra as duas etapas do cálculo do alinhamento ótimo entre as sequências $x = GTGTATACCAGATG$ e $y = GTACCCAG$. A Figura 3.5 exibe um exemplo de alinhamento ótimo obtido a partir do algoritmo de alinhamento local.

	–	G	T	A	C	C	C	A	G
–	0	0	0	0	0	0	0	0	0
G	0	3	2	2	1	0	0	2	3
T	0	2	6	5	4	3	2	1	3
G	0	3	5	4	5	4	3	4	5
T	0	2	6	5	5	6	5	4	5
A	0	2	5	9	8	7	7	8	7
T	0	1	5	8	10	9	8	7	9
A	0	2	4	9	9	11	10	12	11
C	0	1	3	8	12	12	14	13	14
C	0	0	2	7	11	15	15	15	14
A	0	2	1	6	10	14	16	18	17
G	0	3	3	5	9	13	14	18	21
A	0	2	3	7	8	12	14	17	20
T	0	1	5	6	8	11	13	16	19
G	0	3	4	6	7	10	12	15	19

(a)

	–	G	T	A	C	C	C	A	G
–	0	0	0	0	0	0	0	0	0
G	0	3	2	2	1	0	0	2	3
T	0	2	6	5	4	3	2	1	3
G	0	3	5	4	5	4	3	4	5
T	0	2	6	5	5	6	5	4	5
A	0	2	5	9	8	7	7	8	7
T	0	1	5	8	10	9	8	7	9
A	0	2	4	9	9	11	10	12	11
C	0	1	3	8	12	12	14	13	14
C	0	0	2	7	11	15	15	15	14
A	0	2	1	6	10	14	16	18	17
G	0	3	3	5	9	13	14	18	21
A	0	2	3	7	8	12	14	17	20
T	0	1	5	6	8	11	13	16	19
G	0	3	4	6	7	10	12	15	19

(b)

Figura 3.6: (a) Matriz de similaridade (b) Recuperação do alinhamento local ótimo

3.6 Alinhamentos Próximos ao Ótimo

Para muitos pares de sequências os algoritmos global, semi-global e local são capazes de encontrar mais de um alinhamento com a pontuação máxima (considerando um sistema de pontuação ω). Podemos interpretar este fato como a possibilidade de haver mais de uma maneira de alinhar as duas sequências de uma forma considerada "ótima".

Tanto os algoritmos global e semi-global quanto o algoritmo para alinhamento local obtêm, no passo de recuperação do alinhamento, somente um alinhamento ótimo a partir da matriz de similaridades M_s, mesmo que haja outros com a mesma pontuação. Porém, em determinados casos é interessante, e até indispensável, conhecer não somente os demais alinhamentos ótimos como também os alinhamentos com pontuação próxima à ótima.

Alguns pesquisadores desenvolveram soluções para este problema tanto para o alinhamento global quanto para o local [102, 106, 77, 98, 95]. O trabalho de Waterman e Eggert [102] obtém alinhamentos ótimos e próximos ao ótimo entre duas sequências a partir de M_s previamente construída com o algoritmo de alinhamento local.

Para recuperar cada alinhamento ótimo e próximo ao ótimo, os autores procuram em M_s pelo melhor alinhamento (que se inicia na posição da matriz que contém o maior valor e termina numa posição com valor 0). Após recuperá-lo, as posições de M_s "influenciadas" por este alinhamento são recalculadas de modo a eliminar esta influência.

A Figura 3.7 mostra um exemplo de cálculo de alinhamento ótimo na matriz de substituição. A Figura 3.8 mostra as posições na matriz influenciadas pelo alinhamento ótimo a serem recalculadas. Após a redefinição destes valores é encontrado outro alinhamento (próximo a ótimo) numa região da matriz que não intersecta o primeiro alinhamento

encontrado (Figura 3.9).

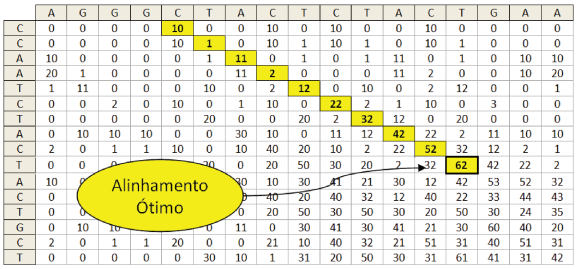


Figura 3.7: Exemplo de matriz em que serão obtidos os alinhamentos ótimos e próximos a ótimo

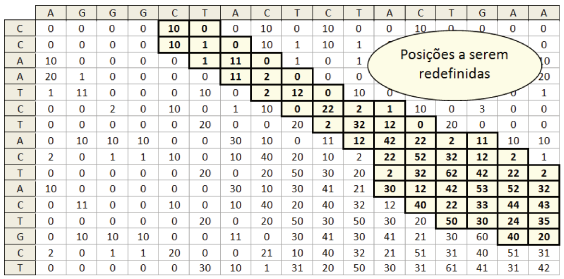


Figura 3.8: Posições influenciadas pelo alinhamento ótimo a serem recalculadas

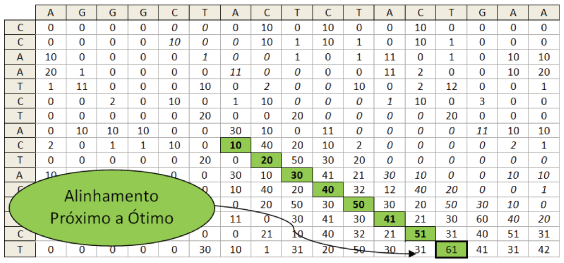


Figura 3.9: Alinhamento próximo a ótimo

O procedimento de recuperação do alinhamento é então repetido k vezes até que os k melhores alinhamentos sejam obtidos.

3.7 Variações no Sistema de Pontuação

Esta seção discute algumas variações feitas no sistema de pontuação utilizado no alinhamento de pares de sequências. Como dito anteriormente, o sistema de pontuação é uma

das entradas para o alinhamento e seus valores influenciam o modo como o algoritmo interpreta o relacionamento entre cada par de símbolos.

3.7.1 Função de Penalidade Afim para Espaços

As operações de indel no alinhamento são, na maioria das vezes, indispensáveis quando as sequências possuem regiões entre as quais há pouca ou nenhuma similaridade. Geralmente, adota-se a função de penalização constante para espaços. Ela é composta por um único valor utilizado para penalizar toda ocorrência de indel. Porém, quando este sistema de pontuação é adotado em alinhamentos onde ocorrem muitas operações de indel seguidas, o custo dos espaços na pontuação total pode se tornar muito alto.

Alguns trabalhos mostram o uso de uma função de penalização para indel que adota o conceito de penalidade "afim" para espaços (do inglês *affine gaps*) [36, 92]. A penalidade afim para espaços é definida considerando-se dois valores:

1. Penalidade para abertura de espaço (v_a): geralmente uma penalidade alta é atribuída para a primeira ocorrência de um espaço no alinhamento;

2. Penalidade para espaços subsequentes (v_s): os espaços subsequentes e contíguos são penalizados de forma atenuada. O valor menor causa menos impacto na pontuação total do alinhamento.

Considerando-se um alinhamento α entre duas sequências x e y temos, para cada bloco de espaços de tamanho k em α, o valor considerado para penalizar os espaços corresponde a $v_a + (k-1) * v_s$ onde v_a é o valor da penalidade de abertura do espaço e v_s é a penalidade (afim) para os espaços subsequentes [36, 51, 21].

Tomando, por exemplo, o alinhamento global de duas sequências x = GTGGCCT-CATCCTATACCAGAG e y = GTACCCAAG, ilustrado na Figura 3.10 com a matriz de pontuação $\mathcal{S}$ da Figura 3.3 - item (b), verificamos que há uma variação significativa na pontuação do alinhamento quando se comparam os resultados obtidos com as funções constante e afim:

- Utilizando a função de pontuação com penalidade constante ($g_1 = -1$): $P_{\alpha\omega_1} = 12$;

- Utilizando o função de pontuação com penalidade de espaços afim ($g_2 = -1 + (k-1)(-0,1)$): $P_{\alpha\omega_2} = 21$.

O maior peso atribuído às operações de indel obtido quando optamos pelo uso da função de penalização constante pode influenciar no resultado do alinhamento ótimo, pois diminui sua pontuação total. O prejuízo fica mais evidente no alinhamento local que não aceita valores negativos, podendo levar inclusive à classificação de um alinhamento

```
GTGGCCTCATCCTATACCAGAG
|             || *||| ||
G-----------TA-CCCA-AG
```

Figura 3.10: Exemplo de alinhamento com muitos espaços

não tão bom como sendo ótimo por conta do alto peso das penalidades atribuídas aos espaços com o uso da função constante.

Outra questão alvo de pesquisas é a determinação dos valores a serem atribuídos a cada penalidade, para qualquer que seja a função de penalização de espaços adotada. Ainda não há consenso sobre quais valores são mais adequados para penalizar os espaços [85].

3.7.2 Matrizes de Pontuação

As matrizes de pontuação (S) também são alvo de estudos que têm o objetivo de melhorar a pontuação para matches e mismatches. Há iniciativas que usam esquemas simples (chamados sistemas de pontuação fixa) em que a matriz se resume a apenas dois valores $S[i,j] = v_{ma}$ para $i = j$ e $S[i,j] = v_{mi}$ para $i \neq j$ com $i,j \in \Sigma$. Neste caso, todo mismatch recebe o mesmo valor idependentemente do grau de similaridade entre os dois símbolos. As matrizes de pontuação utilizadas no alinhamento de sequências de DNA e RNA geralmente utilizam esquemas simples.

A definição das matrizes de pontuação para o alinhamento de sequências de aminoácidos utiliza métodos estatísticos baseados em observações biológicas e dados históricos de evolução entre as sequências [41]. Esta estratégia é utilizada pois as proteínas possuem propriedades bioquímicas que determinam como os aminoácidos são substituídos durante a evolução [79].

As matrizes de pontuação mais utilizadas no alinhamento de sequências de aminoácidos são as matrizes PAM (*Point Accepted Mutations*) [28] e BLOSUM (*BLOcks SUbstitution Matrix*) [44]:

- As matrizes PAM formam um conjunto construído a partir da observação de 1572 mudanças observadas em sequências com mais de 85% de similaridade pertencentes a 71 famílias de proteínas relacionadas. Cada matriz PAM tem 20 linhas e 20 colunas representando os aminoácidos; cada uma das células da matriz representa a probabilidade da substituição de um anminoácido por outro. A matriz PAM1 é calculada a partir da comparação entre seqüências com menos de 1% de divergência e as demais matrizes do mesmo conjunto são calculadas a partir de derivações desta matriz;

- As matrizes BLOSUM são outro conjunto composto a partir de estudos realizados em alinhamentos locais de sequências de proteínas provenientes de 504 famílias. Deste conjunto, a matriz mais utilizada é a BLOSUM62 que contém as pontuações obtidas a partir da análise de alinhamentos de sequências de proteínas com mais de 62% de identidade.

Ao utilizar matrizes como PAM e BLOSUM durante o alinhamento de sequências de proteínas, o algoritmo é capaz de obter resultados que refletem a similaridade do ponto de vista da frequência de mutação entre os aminoácidos diferentemente do que ocorre quando utiliza apenas dois valores simples na matriz de pontuação.

3.8 Considerações Finais

Os algoritmos mostrados nas Seções 3.3, 3.4 e 3.5 apresentam a mesma complexidade em termos de tempo de execução. A complexidade da primeira etapa do algoritmo de alinhamento, no qual é construída a matriz de similaridades, é $O(m * n)$ onde m e n correspondem aos tamanhos das sequências alinhadas. Já, a segunda etapa do algoritmo tem complexidade de tempo igual a $O(m + n)$.

Comparamos os algoritmos de alinhamento de sequências com algoritmos de busca exata, como KMP [53] e Boyer-Moore [16]. A seguir descrevemos as principais diferenças entre os dois tipos de algoritmos:

- Dados dois símbolos $x[i]$ e $y[j]$ com $x[i] \neq y[j]$ e $x[i] \sim y[j]$ ($x[i]$ é semanticamente similar a $y[j]$), os algoritmos de alinhamento de sequências são capazes de "perceber" a similaridade existente entre tais símbolos e, assim, alinhá-los da forma mais adequada. Porém, com a busca exata, o fato dos símbolos serem diferentes já indica que o padrão procurado (no caso, o símbolo $x[i]$) não corresponde ao padrão encontrado (o símbolo $y[j]$), o que resulta numa busca mal-sucedida;

- O algoritmo de alinhamento de sequências fornece, através da visualização do alinhamento ótimo, os pontos de similaridade e de discrepância entre as sequências alinhadas, o que não ocorre (de forma inerente) com os algoritmos de busca exata.

- A matriz de similaridades M_s, na maioria dos casos, não contém apenas um alinhamento ótimo entre as sequências, mas vários alinhamentos que mostram (quando recuperados através de algoritmos como o desenvolvido em [102]) diferentes possibilidades de combinação entre as sequências.

A Tabela 3.3 contém um resumo comparativo dos métodos de alinhamento baseados em programação dinâmica para duas sequências x e y. M_s é a matriz de similaridades com dimensões $m \times n$ utilizada pelos algoritmos para cálculo dos alinhamentos.

Tabela 3.3: Diferenças entre os Algoritmos Apresentados

	Global	Semi-global	Local
Ignorar prefixo	Não	Em x ou y	Em x e y
Ignorar sufixo	Não	Em x ou y	Em x e y
Inicializar com zeros	$M_s[0,0]$	$M_s[i,0]$ e/ou $M[0,j]$	$M_s[i,0]$ e $M_s[0,j]$
Pontuação máxima em	$M_s[m,n]$	$M_s[i,m]; M_s[n,j]$	$M_s[i,j]$

Há várias pesquisas que visam melhorar o desempenho em relação ao tempo de execução e espaço utilizado pelos algoritmos apresentados [76, 49, 10, 48]. Porém, neste trabalho decidimos pesquisar apenas os algoritmos "clássicos" para uso no contexto da avaliação de resultados de testes de robustez. Esta decisão baseou-se na maior facilidade de encontrar informações e implementações de tais algoritmos.

A listagem das duas etapas (construção da matriz de similaridades e recuperação do alinhamento ótimo) de cada um dos algoritmos apresentados nas Seções 3.3, 3.4 e 3.5 encontra-se no Apêndice A.

Capítulo 4

Matriz de Pontuação para Eventos do SUT

Na bioinformática, os sistemas de pontuação são criados através de métodos bastante sofisticados, conforme descrito na Seção 3.7. Tais métodos não se adequam quando nos referimos à criação de sistemas de pontuação para eventos de SUTs pois são baseados em conceitos específicos ao tipo das sequências a serem alinhadas (DNA, RNA e proteínas).

Neste capítulo justificamos a necessidade da criação de um novo método para a elaboração de sistemas de pontuação para o alinhamento de sequências compostas por eventos de SUTs e descrevemos a solução que encontramos para este problema.

Na Seção 4.1 falamos sobre os sistemas de pontuação. A Seção 4.2 detalha o método proposto como solução para o problema de definição do sistema de pontuação para o alinhamento de eventos de SUTs. Na Seção 4.3 descrevemos os sistemas de pontuação adotados em alguns trabalhos relacionados que empregam alinhamento de sequências fora do contexto da bioinformática e a Seção 4.4 contém as considerações finais.

4.1 Introdução

O sistema de pontuação é uma das entradas fornecidas para o algoritmo de alinhamento de sequências, juntamente com o par de sequências a serem alinhadas. A finalidade do sistema de pontuação é fornecer ao algoritmo a pontuação para cada par de símbolos que compõe o alfabeto das sequências envolvidas no alinhamento.

Na bioinformática, há inúmeros trabalhos de pesquisa que tratam da definição do sistema de pontuação mais adequado para cada caso pois sabe-se que não há um conjunto de valores único que seja adequado a todo tipo de alinhamento. Apesar dos algoritmos já serem aplicados há mais de 30 anos, há ainda hoje vários trabalhos que se concentram na definição de matrizes de pontuação e de penalidades para espaços mais adequadas a cada

caso [47, 31].

De uma maneira geral, ao criar os sistemas de pontuação para alinhamentos de proteínas, os pesquisadores da bioinformática se concentram no histórico da evolução dos aminoácidos. Porém, quando tentamos fazer um paralelo entre as sequências biológicas e os sistemas em teste, concluímos que não é possível utilizar o mesmo raciocínio para a criação dos sistemas de pontuação para avaliação dos resultados de teste de robustez pois a evolução dos SUTs não ocorre no nível de eventos, diferentemente do que ocorre com as sequências biológicas. Assim, não podemos construir um sistema de pontuação utilizando as mesmas bases da bioinformática.

Por este motivo, desenvolvemos, como parte desta pesquisa, um novo método para criação do sistema de pontuação para o alinhamento de eventos do SUT. Este método considera a similaridade semântica (ou funcional) entre os eventos para pontuar o alinhamento entre eles.

4.2 Construção da Matriz de Pontuação para cada SUT

No contexto do alinhamento das sequências resultantes da execução de sistemas durante os testes de robustez, cada símbolo σ presente nas sequências é um evento do SUT que pertence ao alfabeto Σ_{ev} dos eventos do sistema. O alfabeto Σ_{ev} tende a ser específico cara cada SUT assim como o grau de relacionamento entre cada um dos eventos de Σ_{ev}. Com isto, podemos afirmar que a utilização de um sistema de pontuação genérico a todos os SUT não é capaz de fornecer ao alinhamento informações reais de como cada par de eventos do SUT se relaciona.

Neste trabalho, mostramos através de estudos de caso (no Capítulo 5), que a qualidade dos alinhamentos realizados utilizando-se sistema de pontuação especificamente desenvolvidos para o SUT é superior à qualidade dos alinhamentos das mesmas sequências realizados com a aplicação de um sistema fixo (matriz de pontuação com apenas um valor para todo match e outro para todo mismatch).

Procuramos, então, elaborar um método para composição do sistema de pontuação, tomando como base o conhecimento sobre o papel de cada um dos eventos do SUT e, assim, definirmos o nível de relacionamento entre eles.

Para dispor os eventos de uma forma que fossem agrupados conforme seu papel no SUT nós nos inspiramos no conceito de *árvore de categorização de eventos* descrito no trabalho de Salfner et al. [90]. Os autores propõem uma árvore para descrever os eventos de forma hierarquizada. A partir dessa árvore é possível obter uma medida de distância entre os eventos. A meta do trabalho citado é melhorar a expressividade de logs de eventos. Os autores dizem que não há um método para a escolha do local mais apropriado a cada evento pois a construção da árvore é específica a cada problema.

Assim, surgiu uma nova questão: como criar a árvore de categorização para a representação do relacionamento semântico entre os eventos do SUT?

Para responder esta questão consultamos um método de geração de casos de teste descrito por Grochmann e Grimm em [37], extensão do método de partição por categoria proposto por Ostrand e Balcer [81]. A idéia central no trabalho de Grochmann e Grimm é a análise do domínio de entrada do sistema e sua divisão em partições. As partições são escolhidas pelo testador para a diferenciação clara das entradas na árvore. As partições devem ser completas (todos os eventos devem estar na árvore) e disjuntas (um mesmo evento não pode estar em mais de uma partição) para evitar ambiguidades no cálculo de distância entre os eventos. Os autores consideraram apenas o domínio de entradas foi considerado, pois a meta é construir casos de teste a partir da árvore.

Adotamos esta ideia e a adaptamos para a definição do sistema de pontuação a ser utilizado no alinhamento de sequências de eventos. Em nosso caso consideramos tanto os eventos de entrada quanto os eventos de saída pois para avaliação de resultados de teste de robustez é importante analisar tanto as entradas inseridas como as saídas apresentadas pelo SUT.

Definimos, então, a criação da árvore de categorização de eventos da seguinte forma:

1. Definimos um nó central (raiz), que representa o alfabeto do SUT $\Sigma = \Sigma_{ev} \cup \Sigma_{er}$, sendo Σ_{ev} os eventos regulares do SUT e Σ_{er} os eventos de erro mapeados a partir da ocorrência durante os testes.

2. Imediatamente abaixo da raíz são descritos os nós superiores representando os conjuntos Σ_{ev} e Σ_{er}.

3. Σ_{ev} é dividido em Σ_{ev_i}, eventos de entrada e Σ_{ev_o}, eventos de saída do SUT.

4. A partir desta divisão inicial segue a etapa específica a cada SUT, na qual todos os eventos são distribuídos na árvore. A meta é que cada nó contenha um agrupamento de eventos diferentes, porém funcionalmente similares. Tais eventos, quando alinhados, representam para o alinhamento o que chamamos de *good mismatches*, isto é, substituições aceitáveis de eventos. Vale salientar a regra, definida por [37], que define que os eventos devem ser divididos em partições completas e disjuntas.

A Figura 4.1 ilustra os passos iniciais da construção da árvore e a Figura 4.2 ilustra um exemplo da árvore de categorização de eventos. Na Figura 4.2, os eventos do SUT formam o alfabeto $\Sigma_{ev} = \{A, B, C, D, E, F\}$ e os eventos de erro percebidos nos traços resultantes da injeção de falhas formam o alfabeto $\Sigma_{er} = \{Abort, Hang\}$. Os eventos do SUT se dividem em eventos de entrada $\Sigma_{ev_i} = \{A, B, C\}$ e em eventos de saída $\Sigma_{ev_o} = \{D, E, F\}$. Podemos afirmar que os eventos A e B são mais similares do que os eventos A e C. O par de eventos de saída (E, F) é mais similar do que os pares (D, E) e (D, F).

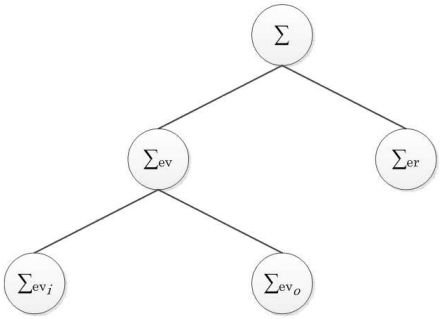


Figura 4.1: Primeiros níveis da árvore de categorização de eventos

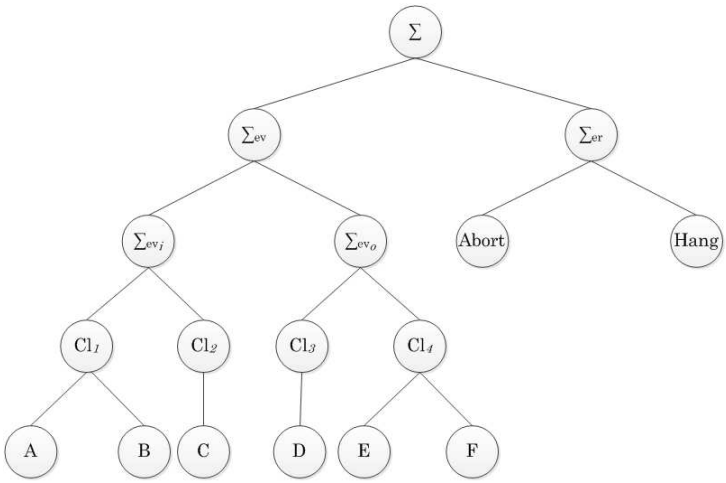


Figura 4.2: Exemplo de árvore de categorização de eventos

Após criarmos a árvore de categorização de eventos podemos perceber todos os eventos do alfabeto Σ como sendo os nós-folha agrupados em partições conforme a similaridade entre eles. Podemos, então, medir a distância entre cada par de nós-folha conforme a Equação 4.1.

$$d(x, y) = \text{número de ramos entre o par de nós-folha } (x, y) \tag{4.1}$$

A distância d obtida entre cada par de eventos $(x, y) \in \Sigma$ é armazenada em uma célula da matriz que denominamos *Matriz de Distâncias* (D). A Tabela 4.1 mostra a matriz de distâncias para a árvore de categorização de eventos da Figura 4.2.

Definimos também a *Matriz de Pontuação* (S) que pode ser preenchida a partir dos resultados da matriz de distâncias. Este é último passo do método para a construção do sistema de pontuação. Cada célula de S é preenchida da seguinte forma: seja $d_{max} =$

Tabela 4.1: Matriz de distâncias D para a árvore da Figura 4.2

-	A	B	C	D	E	F	Crash	Hang
A	0							
B	2	0						
C	4	4	0					
D	6	6	6	0				
E	6	6	6	4	0			
F	6	6	6	4	2	0		
Crash	6	6	6	6	6	6	0	
Hang	6	6	6	6	6	6	2	0

$max\{d(x,y), \forall x,y \in \Sigma\}$ então cada célula em S é calculada segundo a Equação 4.2, assumindo que quanto mais próximos estão dois nós na árvore de categorização de eventos mais similares eles são.

$$s(x,y) = d_{max}(x,y) - d(x,y) \tag{4.2}$$

De acordo com a Equação 4.2, a pontuaçao máxima é dada para matches $S(x,x)$ pois a distância de um evento para ele mesmo na árvore de categorização de eventos é 0 (zero). Os mismatches, isto é, pares (x,y) com $x \neq y$, representam operações de substituição no alinhamento e recebem pontuações variáveis inversas às distâncias entre os nós x e y na árvore.

A Tabela 4.2 corresponde à matriz de pontuação referente à matriz de distância da Tabela 4.1.

Tabela 4.2: Matriz de pontuação S correspondente à matriz de distância da Tabela 4.1

-	A	B	C	D	E	F	Crash	Hang
A	6							
B	4	6						
C	2	2	6					
D	0	0	0	6				
E	0	0	0	2	6			
F	0	0	0	2	4	6		
Crash	0	0	0	0	0	0	6	
Hang	0	0	0	0	0	0	4	6

4.3 Trabalhos Relacionados

Há alguns trabalhos que utilizam algoritmos de alinhamento de sequências fora do contexto da bioinformática [25, 26, 67, 52, 15]. Nesta seção, resumimos suas soluções e discutimos sobre os sistemas de pontuação adotados por eles.

Markov e Kalinin [67] desenvolveram uma pesquisa similar a [25] também utilizando alinhamento de sequência, porém global e local, na detecção de intrusão. O sistema de pontuação adotado por eles penaliza mais fortemente mismatches (valor -3) do que espaços (valor -1). Todo match recebe pontuação igual a 2.

Sodiya et al [93] estendem o trabalho apresentado em [25] modificando o algoritmo de alinhamento semi-global para que este trabalhe com valores variáveis para os espaços. As pontuações para matches e mismatches continuam constantes com os valores 1 e 0 respectivamente. Para que os valores atribuídos aos espaços sejam "calibrados", é necessária uma fase preliminar de treinamento onde sequências sem intrusão são alinhadas a assinaturas de usuários válidos. Os autores afirmam que a taxa de falso-positivos no detecção de intrusão diminuiu de $7,7\%$ para $5,4\%$ com a intrução da fase de calibração do alinhamento.

Em 2003, Coull e colegas [25] utilizam o algoritmo de alinhamento semi-global na detecção de intrusão. A comparação é realizada entre dois tipos de sequências: traços capturados da rede e comportamentos de usuários legítimos (não-atacantes) que os autores chamam de "assinaturas dos usuários". O sistema de pontuação utilizado é bastante simples: matches recebem pontuação 1, mismatches recebem pontuação 0. Os espaços são pontuados diferentemente quando inseridos no traço ou na assinatura do usuário. Todo espaço inserido no traço recebe penalidade igual a -2, enquanto que os espaços inseridos na assinatura do usuário recebem penalidade igual a -3. Os autores justificam a penalidade maior para espaços na assinatura do usuário pelo fato de que os eventos da assinatura devem ser executados na ordem em que são definidos, sem a ocorrência de outros eventos intermediários.

Em 2008, Coull e Szymanski extenderam o trabalho desenvolvido em 2003 [25] e propuseram dois diferentes sistema de pontuação para a comparação da sequência de eventos coletada da rede com a assinatura de um usuário válido para verificar se o usuário pode ser um intruso tentando se passar por um usuário normal [26]. A primeira proposta de sistema de pontuação atribui uma pontuação fixa igual a 2 para matches e, diferentemente do trabalho anterior, usa um único valor para os espaços, sejam eles na assinatura ou no traço avaliado. Ja os mismatches recebem pontuações que variam conforme a funcionalidade dos eventos. Pares de eventos funcionalmente similares quando alinhados não recebem penalização, diferentemente do alinhamento de pares de eventos não relacionados funcionalmente. A segunda proposta consiste na definição de um conjunto restrito

de eventos que o usuário válido sempre executa. Todo alinhamento entre quaisquer dois eventos deste conjunto não é penalizado. Nas duas propostas apresentadas os autores não citam como definem os valores atribuídos aos mismatches mas mostram, através da realização de experimentos, que a primeira proposta mostrou melhores resultados do que a segunda. O trabalho mostra ainda, através de experimentos e comparação dos resultados por meio de análise de curvas ROC [34], que o primeiro sistema de pontuação proposto é melhor do que o sistema de pontuação utilizado em 2003 [25].

Kessentini e colegas [52] utilizam o algoritmo de alinhamento global numa abordagem de teste de transformação de modelos. O objetivo é verificar se os modelos transformados permanecem válidos em relação a um modelo de referência pré-estabelecido. Cada símbolo nas sequências se refere a um conjunto de predicados, conceitos existentes nos modelos como classes ou attributos. A pontuação utilizada no alinhamento entre os modelos transformados e o modelo de referência é a seguinte: todo espaço utilizado pelo algoritmo recebe como penalidade o valor -1. Os mismatches (considerados quando há diferentes predicados no símbolo alinhado) recebem valor 0 e os matches recebem valores variáveis de acordo com o número de parâmetros equivalentes nos predicados.

Bose et al. [15] propuseram uma abordagem que simula o alinhamento múltiplo de um conjunto de sequências de eventos coletados a partir de instâncias de processos de negócios. A abordagem proposta é usada em mineração de processos (*process mining*) com o intuito de verificar a conformidade de tais instâncias, verificar quais os comportamentos comuns e, ainda, identificar os comportamentos excepcionais. Primeiramente, as sequências são agrupadas em clusters através da aplicação do algoritmo AHC (*Agglomerative Hierarchical Clustering*)[27]. Em seguida, as sequências mais semelhantes são alinhadas (em pares) através do alinhamento global. O sistema de pontuação utilizado consiste de apenas dois valores fixos: 1 para matches e -1 para mismatches e espaços.

4.4 Considerações Finais

Realizamos uma pesquisa com o intuito de encontrar trabalhos que aplicassem algoritmos de alinhamento de sequências fora do domínio da bioinformática. Nesta pesquisa pudemos verificar que já há algumas iniciativas para uso de tais algoritmos em áreas como detecção de intrusão. Porém, a maioria faz uso de sistemas de pontuação fixa. Isto se justifica pelo fato de que os trabalhos encontrados são recentes. O único trabalho encontrado que elaborou um sistema de pontuação mais sofisticado [26] conclui, através de experimentos, que tal sistema de pontuação é mais adequado do que o sistema de pontuação fixa utilizado pelos mesmos autores numa fase anterior da pesquisa [25].

Nosso trabalho [61, 62, 63] também apresentou a mesma evolução na definição do sistema de pontuação que verificamos em [25, 26]. Inicialmente, trabalhamos apenas com

sistemas de pontuação fixa (com apenas três valores, para pontuação de matches, mismatches e espaços), então passamos a pontuar diferentemente alguns mismatches, que chamamos de *good mismatches*. Os good mismatches recebiam uma pontuação diferenciada, não tão baixa quanto os demais mismatches (chamados por nós de *bad mismatches)*. Porém, tanto no caso dos sistemas de pontuação constante como nos sistemas com variação de pontuação para mismatches, os valores atribuídos a cada um dos casos era definido de forma empírica, pois não contávamos com um método.

Assim, a definição do método de criação do sistema de pontuação baseado na árvore de categorização de eventos vem completar as abordagens criadas (detalhadas nos dois capítulos subsequentes desta tese).

Alguns autores propuseram ferramentas para ajudar na construção da árvore de categorização de eventos [59, 18] de forma que o testador pode ter auxílio para a criação do sistema de pontuação por meio da automatização de parte da tarefa de criação da matriz de pontuação proposto neste capítulo.

Nos capítulos 5 e 6 apresentamos estudos de caso que utilizam o conceito de árvore de categorização de eventos aqui proposto.

Capítulo 5

Aplicação do Algoritmo de Alinhamento Global

Neste capítulo descrevemos a primeira abordagem desenvolvida para análise de resultados de teste de robustez. Esta abordagem estende a técnica de comparação do traço de execução com o padrão-ouro, comumente utilizada como oráculo [83]. Em nossa solução, o oráculo criado utiliza o algoritmo de alinhamento global de sequências na comparação. Este algoritmo trabalha com busca inexata permitindo algumas diferenças entre a sequência modelo e a sequência observada. Esta abordagem foi chamada de GCSpec (Golden-run Comparison based on the Specification).

A Seção 5.1 introduz a abordagem criada no contexto da avaliação dos resultados do teste de robustez. A Seção 5.2 mostra uma visão geral dos passos da abordagem GCSpec. O detalhamento de cada um dos passos da abordagem é descrito na Seção 5.3. Os estudos de caso realizados como prova de conceito da abordagem são mostrados na Seção 5.4. A Seção 5.5 mostra o uso do algoritmo de alinhamento semi-global numa variação da abordagem GCSpec. Na Seção 5.6 estão descrevemos os trabalhos relacionados e, por fim, na Seção 5.7 estão as considerações finais.

5.1 Introdução

O teste de robustez é uma técnica empregada na verificação do comportamento de um sistema diante de entradas inválidas ou quando este é submetido a condições estressantes. A análise de resultados para este tipo de teste não é trivial porque a documentação do sistema geralmente não engloba o comportamento esperado para entradas inválidas ou para condições fora do "normal".

Atualmente, uma das principais abordagens usada como oráculo para teste de robustez é a comparação com o padrão-ouro, coletado a partir da execução da mesma implementa-

ção do SUT, utilizando-se apenas o workload. O padrão-ouro é apenas uma referência do comportamento que o sistema deveria apresentar caso fosse robusto. Ele não é adequado como modelo exato para o resultado do teste de robustez porque, durante a injeção de falhas, mesmo em caso do SUT ser robusto e determinísta, seu comportamento pode ser diferente do modelo de comportamento normal.

Como afirmamos no Capítulo 2, durante o teste de robustez o SUT é exercitado com o *workload* e também o *faultload*. Nestes casos, quando o SUT é robusto mecanismos de tolerância a falhas podem ser ativados durante os testes tornando trechos do traço de execução resultante diferentes dos trechos correspondentes ao mesmo momento da execução no padrão-ouro, pois durante a coleta do padrão-ouro não há falhas sendo injetadas.

Por este motivo, na comparação com padrão-ouro, a suposição utilizada na comparação com padrão-ouro tradicional de que traços diferente do padrão-ouro apresentam erros leva a uma alta taxa de falso-positivos, mesmo para sistemas determinísticos.

Este capítulo descreve nossa solução que estende o oráculo comumente aplicado na avaliação de traços de execução resultantes da injeção de falhas (comparação com padrão-ouro). Essa extensão é feita através do uso de algoritmos de alinhamento de sequência para comparar traços coletados durante teste de robustez com padrões-ouro.

Conforme apresentamos no Capítulo 3, o algoritmo de alinhamento de sequências apresenta algumas características úteis que serviram de motivação para sua adoção no papel de procedimento do oráculo:

- Pertence ao grupo de algoritmos de matching inexato, permitindo algumas variações entre as sequências a serem alinhadas;
- Utiliza o sistema de pontuação que possibilita quantificar a similaridade entre os símbolos que compõem as sequências envolvidas no alinhamento;
- Provê como resultado, não somente o veredicto sobre o traço em relação ao padrão-ouro, mas também a pontuação entre as sequências alinhadas e a visualização do alinhamento facilitando a identificação dos pontos similares e diferentes entre elas.

5.2 Visão Geral da Abordagem GCSpec

Desenvolvemos a abordagem GCSpec em cinco passos descritos a seguir e ilustrados na Figura 5.1:

1. Obtenção dos conjuntos ρ e τ. O conjunto de padrões-ouro $\rho = \{\rho_1, \rho_2, \ldots, \rho_r\}$ tem tamanho $|r|$ sendo r o número de casos de teste. O conjunto ρ é coletado com o mesmo *workload* aplicado ao teste de robustez porém sem a injeção de falhas. O conjunto de traços de execução coletado durante o teste de robustez

$\tau = \{\tau_{11}, \tau_{12}, \ldots, \tau_{rs}\}$, tem tamanho $|r * s|$ em que r é o número de casos de teste e s é o número de falhas injetadas;

2. Filtragem e codifição dos traços (τ) e dos padrões-ouro (ρ). A filtragem é necessária pois a abordagem GCSpec trabalha com a comparação das informações presentes no fluxo de controle das sequências (eventos de entrada e saída do SUT). Assim, neste passo, toda informação não utilizada pela abordagem (ex: fluxo de dados) é eliminada. Em seguida, é feita a codificação de cada um dos eventos presentes nas sequências para permitir o alinhamento pelo algoritmo e facilitar a visualização dos alinhamentos obtidos;

3. Criação do sistema de pontuação(ω). Na abordagem utilizamos o sistema de pontuação variável, criado conforme o método descrito no Capítulo 4. O sistema de pontuação utiliza aspectos da especificação do SUT para a definição das similaridades entre os pares de eventos possíveis;

4. Alinhamento de cada par de sequências (ρ_i, τ_{ij}) com o algoritmo de alinhamento global para todo $1 \leq i \leq r$ e todo $1 \leq j \leq s$;

5. Obtenção das saídas de cada alinhamento:

   - Alinhamento ótimo ($\alpha_{ot(\omega)}(\rho_i, \tau_{ij})$): possibilita ao testador visualizar os pontos similares (matches) e discrepantes (mismatches e espaços) entre as duas sequências τ_{ij} e ρ_i com relação a um sistema de pontuação ω;
   - Pontuação do alinhamento ótimo ($\wp_{\alpha_{ot(\omega)}}(\rho_i, \tau_{ij})$): pontuação em relação ao número de matches, mismatches e espaços no alinhamento entre as sequências ρ_i e τ_{ij};
   - Similaridade ($\mu_\omega(\rho_i, \tau_{ij})$): porcentagem de similaridade entre um dado par de sequências ρ_i e τ_{ij}. A métrica varia de 0 (quando não há nenhum evento similar entre as sequências e só ocorrem deleções e inserções) até 1 (quando $\tau_{ij} = \rho_i$ e ocorrem apenas matches);
   - Veredicto: julgamento que informa se o traço τ_{ij} passou ou falhou na comparação com o padrão-ouro ρ_i. O veredicto é obtido comparando-se o valor obtido para a métrica de similaridade com um limiar previamente definido.

5.3 Detalhamento dos Passos da Abordagem GCSpec

Nesta seção detalhamos a descrição de cada um dos passos da abordagem GCSpec.

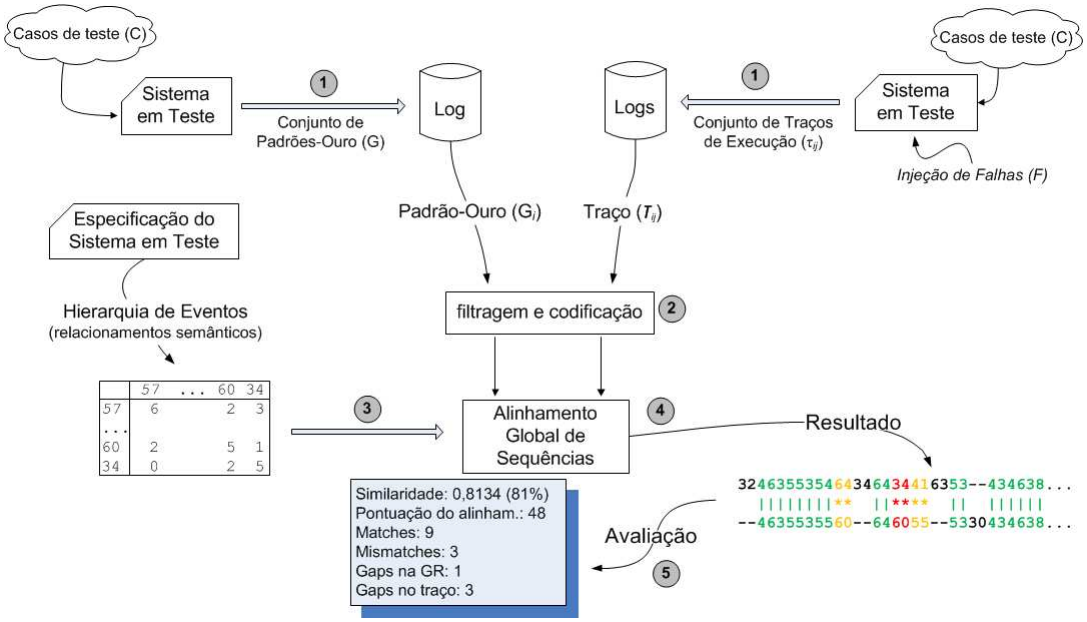


Figura 5.1: Visão geral da abordagem GCSpec

Durante o ciclo de execução dos testes de robustez, para cada sistema a ser testado, são definidos dois conjuntos:

- Um conjunto de casos de teste $C = c_1, ..., c_r$ que forma o *workload* do SUT;
- Um conjunto de falhas $F = f_1, ..., f_s$ a serem injetadas (*faultload*);

A partir da definição dos conjuntos C e F são realizados os testes de robustez através da técnica de injeção de falhas. Para cada caso de teste c_i e cada falha injetada f_j coletamos, durante o teste, um traço τ_{ij}. Ao final do teste de robustez temos o conjunto $\tau = \tau_{11}, ..., \tau_{rs}$. Exercitando o SUT com o mesmo *workload* porém sem a injeção de falhas obtemos o conjunto dos padrões-ouro $\rho = \rho_1, ..., \rho_r$. Os tamanhos dos conjuntos τ e ρ são, respectivamente, $|r * s|$ e $|r|$ em que r é o número de casos de teste presentes em C e s é o número de falhas injetadas presentes em F.

No segundo passo da abordagem, os traços de τ e ρ são preparados para se transformarem nas sequências de entrada para o alinhamento global (descrito no passo 4). Esta preparação consiste em filtrar e codificar todos os elementos de τ e ρ.

Quando coletados, os traços contêm tanto o fluxo de dados como o fluxo de controle resultantes da execução do SUT com o conjunto de casos de teste de C e as falhas de F. Porém, a abordagem GCSpec compara apenas os fluxos de controle dos elementos de ρ e τ. Então, antes de iniciar o alinhamento das sequências é necessário filtrá-las para manter apenas as informações relevantes à abordagem.

Para finalizar a preparação dos traços, codificamos cada um deles de forma a permitir o alinhamento pelo algoritmo de programação dinâmica e, ainda, para facilitar a visualização do alinhamento obtido.

Para isto, definimos uma *Função de Codificação* descrita na Equação 5.1 através da qual codificamos todos os eventos do conjunto $\Sigma = \sigma_1, ..., \sigma_t$, sendo que $\Sigma = \Sigma_{ev} \cup \Sigma_{er}$.

$$C_\Sigma : \Sigma \to \mathbb{N} \tag{5.1}$$

O passo seguinte é a definição do sistema de pontuação ω, terceira entrada para o algoritmo de alinhamento global. O sistema de pontuação é criado para permitir ao algoritmo a formação dos alinhamentos de maneira que representem, da forma mais precisa possível, o grau de similaridade existente entre os símbolos que compõem as sequências. Sistemas de pontuação inadequados levam o algoritmo a considerar erroneamente pares de eventos não relacionados como sendo similares.

Como mostramos no Capítulo 3, o sistema de pontuação apresenta duas partes: a matriz de pontuação (S) e a função de penalidades para ocorrência de espaços (g). A matriz S é construída com base na *Árvore de Categorização de Eventos* apresentada na Seção 4.2, enquanto que a função penalidade de espaços é utilizada de forma constante e recebe, para esta abordagem, o valor fixo $g = -1$. De posse de ω, ρ e τ podemos iniciar o passo 4 em que são realizados os alinhamentos com o algoritmo de alinhamento global.

Para o alinhamento entre cada par de sequências composto por um padrão-ouro ρ_i e um traço τ_{ij} o algoritmo constrói uma matriz de similaridades (M_s). A matriz é montada de acordo com as Equações 3.2, 3.3 e 3.4 descritas na Seção 3.3. Após o cálculo de M_s para o par (τ_{ij}, ρ_i), temos as seguintes saídas da abordagem GCSpec:

- As sequências alinhadas de forma "ótima" ($\alpha_{ot(\omega)}(\rho_i, \tau_{ij})$);
- A pontuação do alinhamento ótimo ($\wp_{\alpha_{ot(\omega)}}(\rho_i, \tau_{ij})$);
- A similaridade ($\mu_\omega(\rho_i, \tau_{ij})$) calculada através da métrica de similaridade proposta (Equação 5.2), que tem por objetivo mensurar a porcentagem do padrão-ouro ρ_i alinhada ao traçoτ_{ij};
- O veredicto sobre a robustez do traço τ_{ij} em relação ao padrão-ouro ρ_i, considerando-se um limiar ℓ pré-definido.

O alinhamento entre duas sequências é organizado comumente em três linhas, conforme o exemplo da Figura 5.2:

- Primeira linha, o padrão-ouro ρ_i codificado de acordo com a Equação 5.1;

- Segunda linha, o símbolo "|" representa o alinhamento de eventos idênticos no padrão-ouro e no traço, e o símbolo "⋆" representa o alinhamento entre dois eventos diferentes;
- Terceira linha, o traço τ_{ij} obtido com a injeção de falhas, também codificado conforme a Equação 5.1.

Na maioria dos alinhamentos podemos perceber a ocorrência de espaços ("−") na primeira e terceira linhas do alinhamento, intercalados aos eventos de ρ_i e τ_{ij}. Os espaços representam a inserção (espaço no padrão-ouro) ou a deleção (espaço no traço) de eventos.

A inserção de um espaço no padrão-ouro corresponde a um evento que não era esperado no padrão-ouro mas que ocorreu quando a falha foi injetada durante o teste de robustez (considerando o mesmo caso de teste c_i). Já, a deleção corresponde à ocorrência de um evento durante a execução de um caso de teste c_i sem a injeção de falhas que não ocorreu quando a falha f_j foi injetada considerando-se também o mesmo caso de teste.

Se colorirmos o alinhamento conforme a similaridade do par de eventos alinhado podemos obter um contraste que serve como mais um auxiliar (além dos símbolos "|" e "⋆") na identificação visual de matches, mismatches e inserções/deleções de eventos.

A Figura 5.2 mostra um exemplo de alinhamento em que cada evento (codificado como um número com dois dígitos) foi colorido conforme o grau de similaride ao evento com o qual foi alinhado. A coloração, neste exemplo, variou do verde - para o alinhamento de pares de eventos idênticos ou muito similares, indo até vermelho - para pares de eventos pouco similares. Inserções e deleções receberam a cor preta.

```
658583--9380698927378921809395169548037895428227589128928589780
||||||  ||||||||||||||||||||||||||**||||||**||||    **********
6585839393806989273789218093951695258037899528 22----939393939340
```

Figura 5.2: Exemplo de alinhamento global entre um padrão-ouro e um traço de execução

A pontuação do alinhamento ótimo é a segunda saída produzida pelo alinhamento global das sequências e consiste da soma das pontuações (ou penalidades) de cada par de eventos de Σ alinhados através de um sistema de pontuação ω. Para o alinhamento da Figura 5.2, calculamos a pontuação considerando o seguinte sistema de pontuação fixa:

- $\omega(i, i) = 7$;
- $\omega(i, j) = 0$ a 4, conforme a similaridade entre i e j, para $i \neq j$;
- $\omega(i, -) = \omega(-, i) = -1$ (penalização constante).

Assim, a pontuação total obtida pelo alinhamento foi $(21 * 7) + (1 * 4) + (1 * 3) + (6 * 0) + (1 * (-1)) + (2 * (-1)) = 151$. Este cálculo corresponde à soma de:

- 21 pares de eventos idênticos (matches) com pontuação 7;
- 8 pares de eventos diferentes (mismatches) com pontuação variando de 4 a 0, de acordo com o grau de similaridade entre os pares de eventos;
- 1 inserção de evento com penalidade -1;
- 2 deleções de eventos com penalidades somando -2.

A pontuação é a quantidade de matches, mismatches e espaços no alinhamento. Porém, através da observação deste número de forma isolada não somos capazes de saber o quão distante o traço de execução analisado está em relação ao padrão-ouro.

Por este motivo desenvolvemos a terceira saída da abordagem que denominamos de *Métrica de Similaridade*. Elaboramos esta métrica para que pudéssemos saber qual a porcentagem de similaridade entre cada traço τ_{ij} em relação ao padrão ρ_i.

Desta forma, ao invés de avaliarmos a pontuação de cada alinhamento de forma isolada podemos "situar" a similaridade do par alinhado dentro de uma escala que varia de 0 a 1, sendo que quanto mais próximo de 1, mais similar é o traço alinhado ao padrão de comportamento. Definimos a métrica de similaridade conforme a Equação 5.2.

$$\mu_\omega(\rho_i, \tau_{ij}) = \frac{min(\wp_{\alpha_\omega}(\rho_i, \tau_{ij})) + \wp_{\alpha_{ot(\omega)}}(\rho_i, \tau_{ij})}{min(\wp_{\alpha_\omega}(\rho_i, \tau_{ij})) + max(\wp_{\alpha_\omega}(\rho_i, \tau_{ij}))} \tag{5.2}$$

em que min($\wp$) é a pontuação mínima possível entre ρ_i e τ_{ij} (situação ilustrada pelo item (a) da Figura 5.3) e max($\wp$) é a pontuação máxima possível entre ρ_i e τ_{ij} (situação ilustrada pelo item (b) da Figura 5.3).

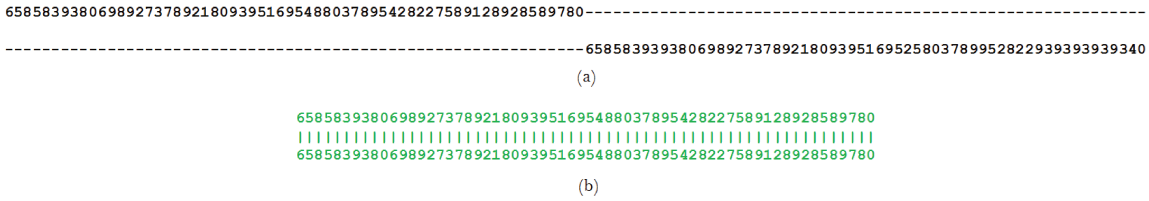


Figura 5.3: Exemplos de pior e melhor alinhamentos entre um padrão-ouro e um traço de execução

A métrica de similaridade nos possibilitou classificar os traços relacionados a um mesmo caso de teste conforme a similaridade entre eles e o padrão-ouro correspondente. Porém, como os valores da métrica são contínuos dentro do intervalo [0,1] ainda não era possível decidirmos se poderíamos considerar um traço como sendo ou não "robusto" com base apenas neste valor.

Então, definimos o veredicto (μ) como última saída da abordagem GCSpec que informa se um traço pode ser ou não considerado como "robusto". O veredicto é calculado em relação ao valor da métrica de similaridade e de um limiar (ℓ) também definido com base em alinhamentos. Desta forma, traços que apresentem $\mu_\omega(\rho_i, \tau_{ij}) > \ell$ são considerados "robustos", caso contrário são considerados "não-robustos".

O cálculo do limiar foi criado de forma similar ao modo utilizado em [26, 1]: para obter ℓ alinhamos todos os padrões-ouro $\rho_i \in \rho$ entre si. Em seguida, obtivemos o valor médio da similaridade ($\bar{\mu}$) para as pontuações obtidas nos alinhamentos entre padrões-ouro. A Equação 5.3 descreve o cálculo para obtenção do limiar.

$$\ell = \bar{\mu}_\omega(\rho_i, \rho_j), 1 \leq i, j \leq r, i \neq j \tag{5.3}$$

5.4 Estudos de Caso

Nesta seção descrevemos o processo de experimentação realizado para verificação da GCSpec. A experimentação foi desenvolvida por meio de estudos de casos com o intuito de verificar se a abordagem GCSpec é adequada para ser utilizada como oráculo estendendo a comparação tradicional com o padrão-ouro.

A experimentação através dos estudos de caso foi realizada conforme proposto em [104]. Realizamos três estudos de caso dividindo o processo em quatro fases:

1. Definição: definição de escopo, objetivo, e foco dos estudos de caso;

2. Planejamento: seleção das variáveis a serem observadas e, com base nestas variáveis, definição do contexto dos estudos de caso. Após a definição do contexto, ocorre a definição do projeto dos estudos de caso.

3. Execução: realização dos testes de robustez e coleta de resultados para análise por meio dos oráculos (abordagem proposta e abordagem existente);

4. Análise e interpretação dos resultados e Conclusão: sumarização e interpretação dos resultados em relação ao objetivo traçado na fase de definição. Comparação dos resultados analisados na fase anterior com as questões levantadas; apresentação das conclusões.

5.4.1 Definição

A definição de um experimento fornece a direção a ser tomada como base para a elaboração das fases seguintes [104].

O propósito deste trabalho é melhorar a abordagem utilizada para verificação de resultados de teste de robustez levando a uma menor taxa de falso-positivos e a resultados (qualitativamente) mais significativos do ponto de vista da similaridade entre as sequências comparadas. Como resultado, a melhoria da abordagem utilizada para verificação de resultados de teste de robustez ocorrerá do ponto de vista do testador e levará, no contexto do teste de robustez, à uma melhor avaliação dos resultados obtidos.

Temos dois objetivos com a realização dos experimentos envolvendo a abordagem GCSpec, proposta e desenvolvida nesta tese. Com base em nossos objetivos definimos as questões a serem respondidas e a métrica a ser utilizada para análise de cada questão.

Objetivo 1: Verificar se o uso da GCSpec é mais adequado do que a comparação com o padrão-ouro tradicional como oráculo para teste de robustez;

- **Q1:** A taxa de falso-positivos na avaliação de resultados de testes de robustez pode ser minimizada adotando-se a GCSpec ao invés da comparação com padrão-ouro tradicionalmente empregada?
- **M1:** O número de resultados errôneamente considerados não-robustos pela comparação com o padrão-ouro é maior do que o obtido pela GCSpec.

Objetivo 2: Verificar no alinhamento de sequências de teste de robustez, se o uso de um sistema de pontuação baseado na especificação leva a melhores resultados quando comparado com o sistema de pontuação fixa.

- **Q2:** O sistema de pontuação variável baseado na especificação do SUT pode guiar o algoritmo de alinhamento global na GCSPec de forma a melhorar a similaridade obtida no alinhamento?
- **M2:** A similaridade obtida no alinhamento quando adotamos o sistema de pontuação variável baseado na especificação do SUT é maior do que a mesma medida obtida com o uso do sistema de pontuação fixa.

5.4.2 Planejamento

Para verificação das questões levantadas foram definidas as seguintes variáveis:

- Variáveis independentes (dados de entrada):
  - Sequências coletadas a partir da execução do SUT sem injeção de falhas (padrões-ouro). Em cada SUT, é gerado um padrão-ouro para cada caso de teste do conjunto $\mathcal{C}$.

  - Sequências coletadas a partir da injeção de falhas (traços resultantes do teste de robustez). Em cada SUT, para cada caso de teste do conjunto $\mathcal{C}$ e cada falha injetada do conjunto $\mathcal{F}$ temos um traço.
  - Sistemas de pontuação: Em cada SUT temos dois sistemas de pontuação a serem aplicados ao alinhamento das sequências: sistema de pontuação fixa e sistema de pontuação variável baseado na especificação do SUT.

- Variáveis dependentes (dados calculados):
  - Taxa de falso-positivos: para cada SUT, é o número de traços errôneamente classificados como não-robustos pela comparação com o padrão-ouro tradicional e pela GCSpec.
  - Similaridade: para os traços avaliados em cada SUT, é o valor obtido pela métrica de similaridade, de acordo com a Equação 5.2.

O experimento foi planejado de forma a utilizar um conjunto de estudos de caso (descritos na Tabela 5.1), para a verificação das questões levantadas em relação à avaliação de resultados de teste de robustez.

Tabela 5.1: Estudos de caso realizados

Estudos de Caso	Traços Obtidos	Casos de Teste	Falhas Injetadas
Elevator	3220	20	160
Cruise Control	2900	25	116
WTP	8	1	8

Os sistemas Elevator e Cruise Control foram obtidos num banco de dados de *benchmarks*[1], juntamente com sua documentação, casos de teste e falhas a serem injetadas. Para estes estudos de caso trabalhamos com injeção de falhas estáticas, ou seja, as falhas foram injetadas no código-fonte do SUT Fem tempo de compilação e o sistema executado com cada uma das falhas injetadas.

Os traços de execução do protocolo WTP foram obtidos a partir de um experimento realizado em outro trabalho de pesquisa [19]. No trabalho citado houve injeção de falhas dinâmica, ou seja, as falhas foram injetadas em tempo de execução através de um injetor de falhas externo.

A seguir, na Seção 5.4.3 descrevemos a execução de cada um dos estudos de caso da Tabela 5.1. Aplicamos os passos da abordagem GCSpec para cada um deles, considerando não apenas o sistema de pontuação variável mas também o sistema de pontuação fixa. O

[1]Software-artifact Infrastructure Repository - http://sir.unl.edu/php/index.php

sistema de pontuação fixa contou com os valores $v_{ma} = 4$, $v_{mi} = 1$ e $g = -1$ para todos os estudos de caso.

Paralelamente ao processo de alinhamento, os traços foram submetidos a dois outros processos:

- Todos os traços de cada SUT foram comparados com os padrões-ouro através da comparação com o padrão-ouro tradicional, conforme abordagem descrita em [83];
- Todos os traços de cada SUT foram manualmente classificados como "robustos" ou "não-robustos".

Por questão de espaço não foi possível apresentar nesta tese todos os resultados obtidos com os estudos de caso. O registro completo pode ser encontrado em nossa página[2].

5.4.3 Execução

Esta seção foi subdividida para melhor descrição do modo como aplicamos a GCSpec em cada estudo de caso.

Elevator

"Elevator" é o nome dado a um sistema utilizado como estudo de caso que simula um controlador de n_e elevadores num edifício com n_f andares. O funcionamento do sistema consiste em obedecer às seguintes requisições:

- Subida (*RequestUp*): O usuário (fora do elevador) requisita um elevador para realizar a ação de subida. Neste caso, o sistema seleciona o elevador mais próximo e o envia para o andar solicitado;
- Descida (*RequestDown*): O usuário (fora do elevador) requisita um elevador para realizar a ação de descida. Neste caso, o sistema seleciona o elevador mais próximo e o envia para o andar solicitado;
- Parada (*RequestStop*): O usuário (dentro do elevador) seleciona um dos andares disponíveis e o elevador segue para o andar solicitado.

Para maiores detalhes sobre o funcionamento do sistema Elevator, pode-se consultar o modelo de estados disponível no Apêndice B.

Primeiramente, executamos o SUT para cada um dos casos de teste disponíveis no conjunto de casos de teste $\mathcal{C}$ sem injetarmos falhas. Cada traço coletado neste passo

[2] www.ic.unicamp.br/~glemos/Resultados/

passou a fazer parte do conjunto de padrões-ouro ρ_{elev}. O tamanho do conjunto ρ_{elev} correspondeu ao número de casos de teste do conjunto $\mathcal{C}$, ou seja, $|\rho_{elev}| = 20$.

Em seguida, foram inseridas 160 falhas no código do SUT. Cada falha, presente no conjunto de falhas $\mathcal{F}$, simulava uma falha real de implementação. Desta forma, o sistema foi exercitado com cada caso de teste de $\mathcal{C}$ para cada uma das falhas de $\mathcal{F}$. Os traços foram coletados formando o conjunto $\tau_{elev} = \tau_1, \ldots, \tau_{3220}$.

Em seguida, todos os traços de execução dos conjuntos ρ_{elev} e τ_{elev} foram filtrados e, os traços resultantes, codificados de acordo com a função de codificação apresentada na Equação 5.1. Os códigos foram atribuídos a cada um dos eventos do SUT e ainda, aos eventos de erro presentes nos traços. A tabela com todos os eventos e seus respectivos códigos está descrita no Apêndice B.

Com base na especificação e no modelo de estados do SUT, definimos a árvore de categorização de eventos (Figura 5.4). Não mostramos todos os nós-folha referentes aos eventos internos por questão de espaço. Os números presentes abaixo dos nomes dos eventos são os códigos atribuídos durante a codificação.

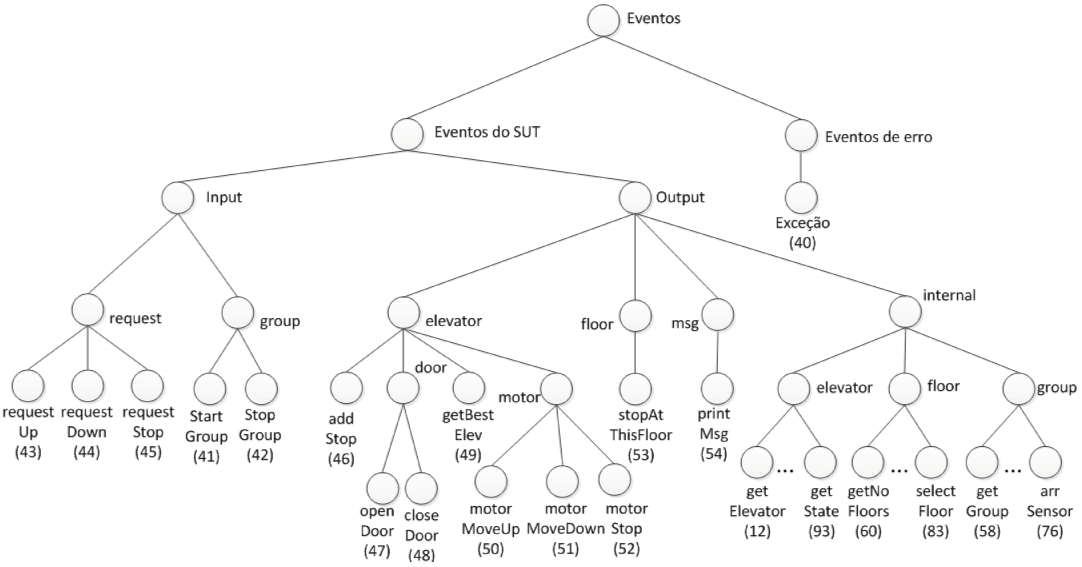


Figura 5.4: Árvore de categorização de eventos do estudo de caso Elevator

A partir das distâncias entre os nós da árvore de categorização de eventos (calculados conforme a Equação 4.1) elaboraramos as matrizes de distância D_{elev} e de pontuação S_{elev}, respectivamente representadas nas Figuras 5.5 e 5.6. A matriz de pontuação S_{elev} e o valor constante da função de penalidade de espaços ($g_{elev} = -1$) formaram o Sistema de Pontuação Variável (SPV_{elev}) utilizado neste estudo de caso.

	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	12	93	60	83	58	76
40	0																				
41	6	0																			
42	6	2	0																		
43	6	4	4	0																	
44	6	4	4	2	0																
45	6	4	4	2	2	0															
46	6	6	6	6	6	6	0														
47	7	7	7	7	7	7	3	0													
48	7	7	7	7	7	7	3	2	0												
49	6	6	6	6	6	6	2	3	3	0											
50	7	7	7	7	7	7	3	4	4	3	0										
51	7	7	7	7	7	7	3	4	4	3	2	0									
52	7	7	7	7	7	7	3	4	4	3	2	2	0								
53	6	6	6	6	6	6	4	5	5	4	5	5	5	0							
54	6	6	6	6	6	6	4	5	5	4	5	5	5	2	0						
12	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	0					
93	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	2	0				
60	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	4	4	0			
83	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	4	4	2	0		
58	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	4	4	4	4	0	
76	7	7	7	7	7	7	5	6	6	5	6	6	6	5	5	4	4	4	4	2	0

Figura 5.5: Matriz de distâncias D_{elev} do estudo de caso Elevator, obtida a partir da árvore de categorização de eventos

	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	12	93	60	83	58	76
40	7																				
41	1	7																			
42	1	5	7																		
43	1	3	3	7																	
44	1	3	3	5	7																
45	1	3	3	5	5	7															
46	1	1	1	1	1	1	7														
47	0	0	0	0	0	0	4	7													
48	0	0	0	0	0	0	4	5	7												
49	1	1	1	1	1	1	5	4	4	7											
50	0	0	0	0	0	0	4	3	3	4	7										
51	0	0	0	0	0	0	4	3	3	4	5	7									
52	0	0	0	0	0	0	4	3	3	4	5	5	7								
53	1	1	1	1	1	1	3	2	2	3	2	2	2	7							
54	1	1	1	1	1	1	3	2	2	3	2	2	2	5	7						
12	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	7					
93	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	5	7				
60	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	3	3	7			
83	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	3	3	5	7		
58	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	3	3	3	3	7	
76	0	0	0	0	0	0	2	1	1	2	1	1	1	2	2	3	3	3	3	5	7

Figura 5.6: Matriz de pontuação S_{elev} obtida a partir da matriz D_{elev}

Conforme descrito na Equação 5.3, para obtermos o veredicto sobre a robustez de um traço τ_{ij} em relação a um padrão ρ_i calculamos um limiar ℓ a partir do qual classificamos os traços alinhados como "robustos" ou "não-robustos". O valor de ℓ é influenciado pelo sistema de pontuação utlizado nos alinhamentos entre os padrões-ouro. Por este motivo, para o estudo de caso Elevator o processo foi repetido para os sistemas de pontuação SPV_{elev} e SPF_{elev}. Os limiares obtidos foram: $\ell_{SPV} = 0,80$ e $\ell_{SPF} = 0,72$.

A Tabela 5.2 resume os critérios utilizados para classificação dos traços alinhados no estudo de caso Elevator.

Nos Exemplos 1 e 2 a seguir, os alinhamento com três linhas seguem o formato: a primeira linha corresponde ao padrão-ouro, a segunda linha contém os símbolos "|" e "⋆" que indicam, respectivamente, as posições em que houve matches e mismatches e,

Tabela 5.2: Limiares para classificação dos traços no estudo de caso Elevator

Sistema de Pontuação	Valor de μ	Veredicto
Fixo	$> 0{,}72$	robusto
	$\leq 0{,}72$	não robusto
Variável	$> 0{,}80$	robusto
	$\leq 0{,}80$	não robusto

finalmente, a terceira linha corresponde ao traço coletado no teste de robustez com injeção de falhas. Esses exemplos ilustram algumas diferenças percebidas entre os alinhamentos realizados com os dois diferentes sistemas de pontuação.

A Figura 5.7 mostra o Exemplo 1 onde trechos de traços do Elevator foram alinhados com os sistemas de pontuação SPF_{elev} e SPV_{elev}. Na coluna 10, as linhas 1-3 mostram um mismatch entre os eventos 69 (`getFloor`) e 34 (`motorStop`).

Ambos são eventos de saída mas 69 é um evento interno e 34 é um evento externo do grupo dos eventos relacionados ao motor do elevador. O sistema de pontuação fixa não tem conhecimento sobre o significado semântico de cada evento e os combina sem considerar este aspecto.

Na coluna 05 da Figura 5.7, as linhas 4-6 mostram um mismatch entre os eventos 30 (`motorMoveUp`) e 34 (`motorStop`); ambos são semanticamente próximos. O mesmo ocorre entre os eventos 71 e 52 da coluna 11 - linhas 1-3, em relação à coluna 06, linhas 4-6 que onde se encontra a melhor combinação entre os eventos 50 e 52.

Figura 5.7: Exemplo 1 - trecho do alinhamento entre um padrão-ouro e um traço com os sistemas de pontuação SPF_{elev} e SPV_{elev}

No Exemplo 2, mostramos o alinhamento completo entre um padrão-ouro ρ_A, que descreve o comportamento esperado para um caso de teste denominado A, e um traço τ_{AB}, que descreve o comportamento do SUT durante o teste de robustez para o caso de teste A com a falha B injetada. O caso de teste A apresenta duas requisições de parada: a primeira requisição deve ser obedecida normalmente pois o andar selecionado é válido. Porém, o segundo andar selecionado é inválido e, portanto, a segunda requisição não deve ser obedecida (neste caso o elevador deve

permanecer no andar atual e emitir uma mensagem de erro).

A falha injetada no código-fonte do sistema alterou o comportamento do elevador. O traço τ_{AB} reflete este comportamento errôneo no qual o sensor que verifica o andar atual do elevador não contém a identificação correta do andar onde o elevador se encontra.

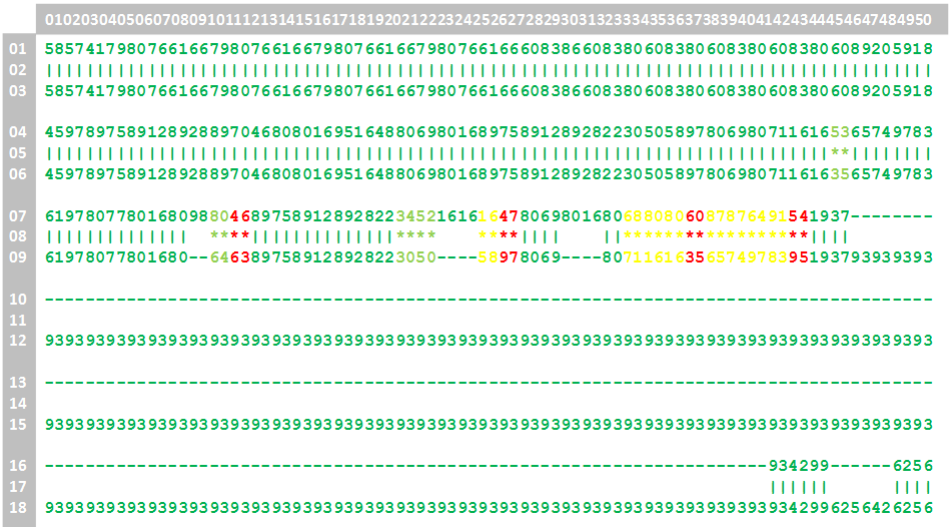


Figura 5.8: Exemplo 2 - alinhamento entre ρ_A e τ_{AB} com o sistema de pontuação SPV_{elev}

A Figura 5.8 mostra o alinhamento entre ρ_A e τ_{AB} com o sistema de pontuação SPV_{elev}. Para este exemplo, a pontuação do alinhamento foi $\wp_{\alpha_{ot(SPV)}}(\rho_A, \tau_{AB}) = 664$. O valor da métrica de similaridade foi $\mu_{SPV}(\rho_A, \tau_{AB}) = 0,80$ e o traço foi considerado, conforme a Tabela 5.2, como sendo não-robusto.

Já, a Figura 5.9 mostra o alinhamento entre ρ_A e τ_{AB} com o sistema de pontuação fixa (SPF_{elev}). Neste caso, o alinhamento obteve pontuação $_{\alpha_{ot(SPF)}}(\rho_A, \tau_{AB}) = 322$. O valor da métrica de similaridade foi $\mu_{SPF}(\rho_A, \tau_{AB}) = 0,78$ e o traço foi considerado como sendo robusto, conforme a Tabela 5.2.

Por questão de espaço quebramos o alinhamento das figuras 5.8 e 5.9 em seis trechos com três linhas cada um.

Além do veredicto ser diferente para os sistemas de pontuação variável e fixa, onde obtivemos um veredicto com verdadeiro positivo para o SPV_{elev} e falso-negativo para o SPF_{elev} podemos notar também nas Figuras 5.8 e 5.9 que a qualidade do alinhamento com o sistema de pontuação variável foi superior ao fixo neste caso pelo número de matches no alinhamento. A Tabela 5.3 resume os números dos alinhamentos das Figuras 5.8 e 5.9.

Cruise Control

"Cruise Control" é um sistema que simula o comportamento do software embarcado que controla automaticamente a velocidade do veículo sem que o motorista tenha que acelerar ou frear. Neste sistema o usuário interage através de alguns comandos básicos:

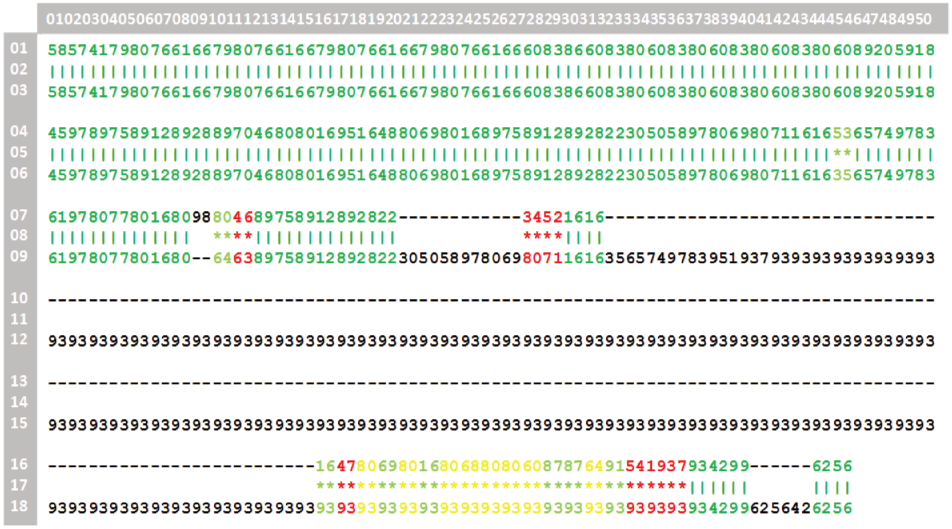


Figura 5.9: Exemplo 2 - alinhamento entre ρ_A e τ_{AB} com o sistema de pontuação SPF_{elev}

Tabela 5.3: Comparação dos números obtidos nos alinhamentos das Figuras 5.8 e 5.9

Sistema de Pontuação	Matches	Mismatches			Espaços	
		Bom	Médio	Ruim	Inserção	Deleção
Fixa	106	8	8	7	124	1
Variável	109	4	8	4	128	5

- Liga/desliga o motor do veículo;
- Liga/desliga o controle de cruzeiro do veículo;
- Acelera/freia o veículo.

O sistema conta com um conjunto $\mathcal{C}_{cruise}$ com 25 casos de teste que foram executados gerando o conjunto ρ_{cruise} com 25 padrões-ouro. Há, ainda, um conjunto $\mathcal{F}_{cruise}$ com 116 falhas estáticas que foram individualmente inseridas no código-fonte simulando falhas de implementação. A partir da injeção de cada uma das falhas $f_i \in \mathcal{F}_{cruise}$ no código, os casos de teste foram exercitados gerando o conjunto τ_{cruise} com 2900 traços de execução.

A árvore de eventos para o Cruise Control encontra-se na Figura 5.10. O modelo de estados e a tabela com a codificação de cada um dos eventos do Cruise Control encontram-se no Apêndice B.

A matriz de pontuação S_{cruise} é ilustrada na Figura 5.11.

Os traços deste estudo de caso são, em média, menores do que os traços do estudo de caso anterior (Elevator). O tamanho médio dos padrões-ouro no Elevator é de 160 eventos. Já, no Cruise Control este número cai para 11 eventos. Os limiares obtidos foram: $\ell_{SPV} = 0,75$ e $\ell_{SPF} = 0,66$.

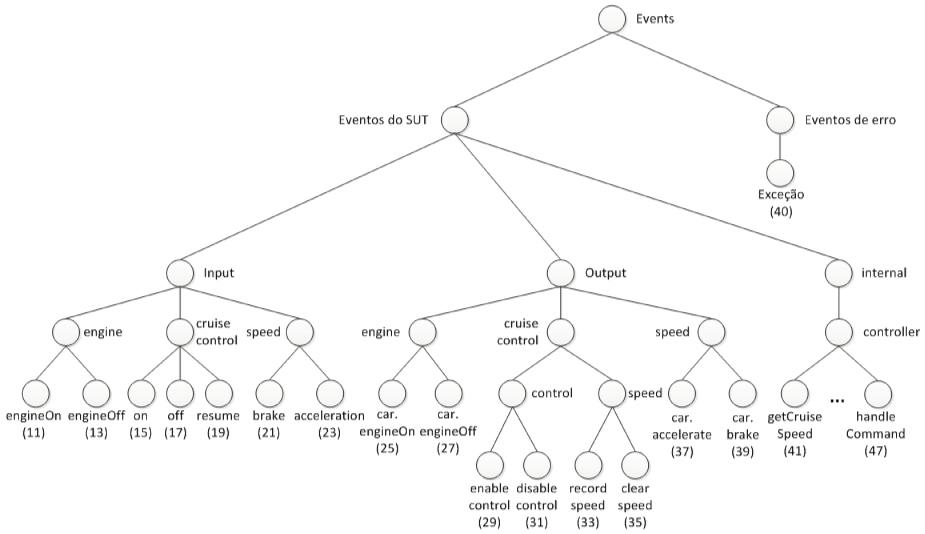


Figura 5.10: Árvore de categorização de eventos do estudo de caso Cruise Control

	11	13	15	17	19	21	23	25	27	29	31	33	35	37	39	41 (*)	47	40
11	7																	
13	5	7																
15	3	3	7															
17	3	3	5	7														
19	3	3	5	5	7													
21	3	3	3	3	3	7												
23	3	3	3	3	3	5	7											
25	1	1	1	1	1	1	1	7										
27	1	1	1	1	1	1	1	5	7									
29	0	0	0	0	0	0	0	2	2	7								
31	0	0	0	0	0	0	0	2	2	5	7							
33	0	0	0	0	0	0	0	2	2	3	3	7						
35	0	0	0	0	0	0	0	2	2	3	3	5	7					
37	0	0	0	0	0	0	0	3	3	2	2	2	2	7				
39	0	0	0	0	0	0	0	3	3	2	2	2	2	5	7			
41(*)	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	7		
47	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	5	7	
40	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	7

Figura 5.11: Matriz de pontuação S_{cruise}

WTP - Wireless Transaction Protocol

O WTP [99] é um protocolo para transações assíncrono, que atua como uma camada do WAP (Wireless Application Protocol) [100]. O WTP oferece três classes de serviços que variam de 0 a 2:

- Classe 0: o processo que começa a transação é chamado Initiator; quem recebe é chamado Responder. Nesta classe, nem o Initiator nem o Responder recebem confirmação durante a transação;
- Classe 1: apenas o Responder recebe confirmação;
- Classe 2: as requisições da transação precisam de confirmação em ambas as direções (Initiator e Responder).

As mensagens vindas da camada mais alta do protocolo são chamadas de primitivas de serviço e seguem o formato `TR-nome.tipo(parametros)`, onde:

- `TR` designa a camada que está provendo o serviço (que é o WTP neste caso);
- `nome` indica o nome da primitiva:
  - invoke: inicia uma transação;
  - result: retorna o reultado de uma transação iniciada anteriormente;
  - abort: aborta uma transação existente;
- `tipo` indica os tipos das primitivas que podem ser:
  - req: indica uma requisição de serviço feita por uma camada superior;
  - ind: indica ao requisitante do serviço uma atividade relacionada à camada que está provendo o serviço;
  - res: resposta a uma primitiva "ind";
  - cnf: confirma que uma requisição foi finalizada com sucesso.
- `parametros`: valores específicos utilizados em cada mensagem como, por exemplo, o Id (identificador) da mensagem.

A análise dos resultados foi realizada em traços coletados em um experimento de injeção de falhas dinâmicas para teste de robustez do WTP classe 2 [19]. Os traços coletados apresentam, em média, 486 eventos.

Pelo fato de haver apenas um padrão-ouro não foi possível calcular um limiar de acordo com o método empregado nos estudos de caso Elevator e Cruise Control. Para o cálculo do limiar precisaríamos de mais de um padrão-ouro.

A abordagem, neste estudo de caso, foi utilizada até o passo de obtenção da métrica de similaridade (μ). Estes passos foram suficientes para classificarmos os traços conforme a similaridade em relação ao padrão-ouro disponível.

A Figura 5.12 mostra a árvore de categorização de eventos do WTP. A tabela com os códigos utilizados na etapa de codificação dos traços encontra-se no Apêndice B.

5.4.4 Sumarização dos Resultados e Conclusões

Como definimos na Seção 5.4.1, os objetivos da experimentação por meio dos estudos de caso foram os seguintes:

1. Validar os resultados em termos de falso-positivos no alinhamento dos traços do conjunto τ com os padrões-ouro do conjunto ρ comparando a GCSpec e a abordagem de comparação com padrão-ouro tradicional.

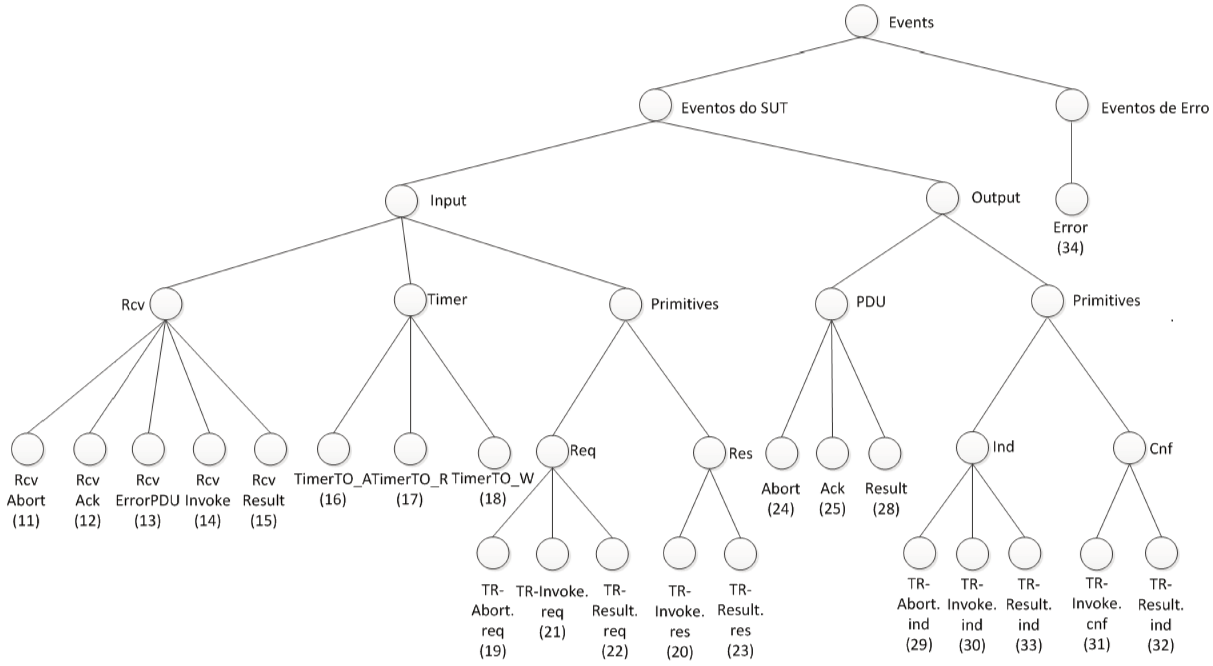


Figura 5.12: Árvore de categorização de eventos do estudo de caso WTP

2. Validar o sistema de pontuação obtido usando o método descrito no Capítulo 4 (chamado de Sistema de Pontuação Váriável ou SPV) em comparação com o sistema de pontuação fixa (SPF) - descrito na Seção 3.7.2 do Capítulo 3. Nos estudos de caso desta tese o SPF foi elaborado com três valores: um valor para matches ($v_{ma} = 4$), um valor para mismatches ($v_{mi} = 1$) e a penalidade para a ocorrência de espaços ($g = -1$).

Em todos os estudos de caso alinhamos os traços dos conjuntos τ e ρ com a GCSpec (SPV) e também com o SPF. Além disso, classificamos todos os traços manualmente e, ainda, os avaliamos através da comparação com padrão-ouro tradicional no qual todo traço minimamente diferente do padrão é considerado como "não-robusto".

A Tabela 5.4 condensa os resultados obtidos para o sistema Elevator. Os limiares considerados para o alinhamento com SPF e para a GCSpec foram, respectivamente, 0,72 e 0,80.

Tabela 5.4: Resumo dos resultados obtidos no estudo de caso Elevator

Elevator	VP	FP	FN	VN
Matching exato	1596	355	0	1269
Alinhamento com SPF	1442	154	12	1612
GCSpec	1558	38	46	1578

Legenda:

- VP (verdadeiro-positivos): a observação manual verificou que o traço apresentava não-robustez e o oráculo verificou não-robustez;
- FP (falso-positivos): a observação manual verificou que o traço não apresentava sinal de não-robustez e o oráculo, errôneamente, verificou não-robustez;
- FN (falso-negativos): a observação manual verificou que o traço apresentava sinal de não-robustez e o oráculo, errôneamente, não verificou sinal de não-robustez;
- VN (verdadeiro-negativos): a observação manual verificou que o traço não apresentava sinal de não-robustez e o oráculo também verificou que não havia sinal de não-robustez;

A Tabela 5.5 apresenta os resultados obtidos com o Cruise Control. Os limiares considerados para o alinhamento com SPF e para a GCSpec foram, respectivamente, 0,66 e 0,75.

Tabela 5.5: Resumo dos resultados obtidos no estudo de caso Cruise Control

Cruise Control	VP	FP	FN	VN
Matching exato	1024	51	0	1825
Alinhamento com SPF	0	0	51	2849
GCSpec	0	0	51	2849

Nos alinhamentos do estudo de caso Cruise Control não houve nenhuma ocorrência em que a similaridade das sequências alinhadas ficou abaixo dos limiares definidos. Isto significa que o alinhamento global, com os dois sistemas de pontuação, não classificou nenhum traço como sendo "não-robusto".

Para o WTP, não obtivemos as taxas de FN, VN, FP e VP por não termos mais de um padrão-ouro para o cálculo do limiar. Apresentamos, nas tabelas 5.6 e 5.7 o resumo dos resultados obtidos com os sistemas de pontuação fixa (SPF_{WTP}) e variável (SPV_{WTP}) nos cinco traços deste estudo de caso.

Tabela 5.6: Resumo dos resultados do WTP com ω variável

τ	$\lvert\tau\rvert$	Sistema de Pontuação Variável (SPV_{WTP})						
		Matches	Mismatches	Inserções	Deleções	$\wp$	μ	$\lvert\alpha\rvert$
1	927	219	80	628	5	1391	0,715806716	933
2	300	256	20	24	28	2053	0,87516469	329
3	10	10	0	0	294	-214	0,036416606	305
4	535	251	45	239	8	1897	0,836441455	544
5	660	256	39	361	9	1805	0,815371025	670

Legenda:

- τ é o traço analisado e $\lvert\tau\rvert$ é o comprimento do traço (em número de eventos);

Tabela 5.7: Resumo dos resultados do WTP com ω utilizando pontuação fixa

τ	Eventos	Sistema de Pontuação Fixa (SPF_{WTP})						
		Matches	Mismatches	Inserções	Deleções	$\wp$	μ	$\lvert\alpha\rvert$
1	927	217	85	625	2	326	0,63628	930
2	300	253	26	21	25	993	0,87747	326
3	10	10	0	0	294	-254	0,03921	305
4	535	250	46	239	8	799	0,79708	544
5	660	253	45	358	6	689	0,75825	667

- $\wp$ é a pontuação do alinhamento entre a sequência τ e o padrão-ouro disponível;
- μ é a porcentagem de similaridade entre um padrão-ouro e um traço τ;
- $|\alpha|$ é o comprimento do alinhamento entre as sequências.

Sumarizamos os resultados dos estudos de caso e obtivemos as conclusões sobre as questões levantadas.

Q1 diz que não há diferença entre as taxas de falso-positivos obtidas com a comparação com o padrão-ouro tradicional e com a GCSpec. Nos estudos de caso Elevator e Cruise Control pudemos verificar que a GCSpec obteve melhores resultados do que o matching exato. Salientamos que os resultados das Tabelas 5.4 e 5.5 dizem respeito a limiares específicos.

A adoção de limiares específicos limita o poder de avaliação da abordagem. Por este motivo, estendemos a comparação tomando o resultado geral da GCSpec através da comparação das curvas ROC (*Receiver Operating Characteristics*) [34] para os estudos de caso Elevator e Cruise Control. Assim, chegamos aos resultados ilustrados na Figura 5.13:

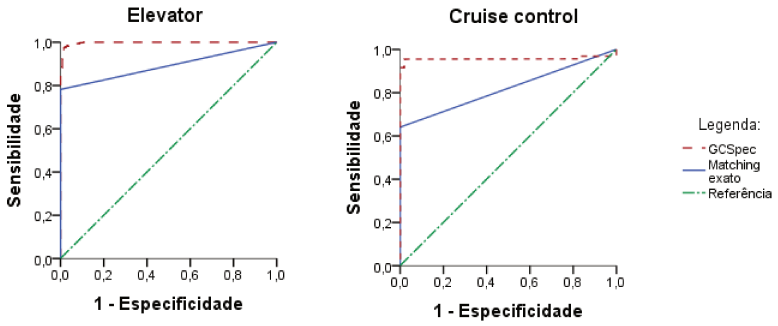


Figura 5.13: Curvas ROC - Comparação entre Matching Exato e a GCSpec

A curva ROC é um gráfico com duas dimensões em que o eixo x representa "1 - especificidade", ou seja, a taxa de falso-positivos e o eixo y representa a taxa de verdadeiro-positivos, ou seja, a sensibilidade. A linha diagonal representa o modelo de comportamento aleatório de um

classificador. Portanto, classificadores que se posicionam acima da diagonal apresentam resultados melhores do que os aleatórios. Detalhes sobre curvas ROC são apresentados no Apêndice C.

A comparação entre curvas ROC de classificadores fornece uma maneira de compará-los independentemente da adoção de um limiar específico. Para isto, é preciso calcular a área sob cada uma das curvas. Quanto maior a área (no intervalo [0,1]) melhor é o classificador. A Tabela 5.8 resume as áreas das curvas da Figura 5.13 para avaliarmos a comparação com padrão-ouro tradicional e a GCSpec para os estudos de caso Elevator e Cruise Control.

Tabela 5.8: Comparação entre a GCSpec e a comparação tradicional com o padrão-ouro: áreas sob as curvas ROC dos estudos de caso Elevator e Cruise Control

	GCSpec	Matching Exato
Elevator	0,997	0,891
Cruise Control	0,957	0,820

Com os resultados obtidos através da medição das áreas sob as curvas ROC para os estudos de caso Elevator e Cruise Control confirmamos Q1.

Em relação à Q2, também realizamos a medição através da área das curvas ROC para os estudos de caso Elevator e Cruise Control. A Figura 5.14 ilustra as curvas ROC e a Tabela 5.9 detalha as áreas sob as curvas.

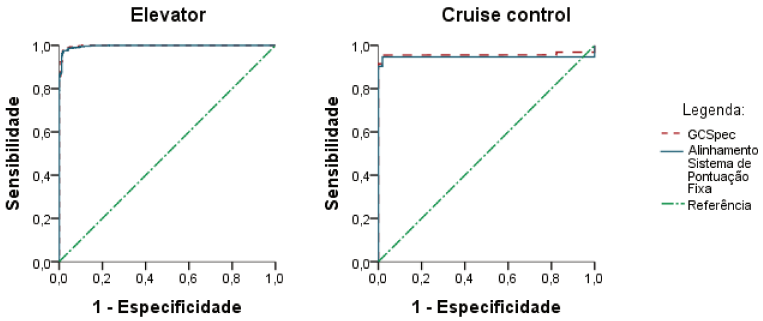


Figura 5.14: Curvas ROC - Comparação entre GCSpec e Alinhamento com sistema de pontuação fixa

Tabela 5.9: Comparação entre a GCSpec e o alinhamento global com sistema de pontuação fixa - Áreas sob as curvas ROC para os estudos de caso Elevator e Cruise Control

	GCSpec	Sist. Pont. Fixa
Elevator	0,997	0,996
Cruise Control	0,957	0,946

No caso de Q2, não estamos diretamente interessados no número de falso-positivos mas na qualidade do alinhamento. Q2 questiona se o sistema de pontuação variável (utilizado na GCSpec) gera alinhamentos com mais qualidade do que o sistema de pontuação fixa. Podemos verificar que os valores obtidos pela GCSpec nos experimentos com os estudos de caso Elevator e Cruise Control foram ligeiramente melhores do que os valores obtidos pelo alinhamento com o sistema de pontuação fixa mas este resultado não nos informa diretamente a respeito da qualidade.

Para ajudar a esclarecer os resultados obtidos realizamos a análise quantitativa no estudo de caso WTP. Nas tabelas 5.6 e 5.7 podemos verificar que o traço mais próximo do padrão-ouro ρ é o traço τ_2. Através da visualização do alinhamento, verificamos que apenas este traço pode ser considerado como "robusto". Todos os demais traços apresentam sinais de não-robustez. Assim, os dados obtidos nos alinhamentos confirmam a observação manual.

Outra característica a observar é que os alinhamentos dos quatro maiores traços τ_1, τ_2, τ_4 e τ_5 obtiveram mais matches com o sistema de pontuação variável. Além disso, de uma maneira geral, o sistema de pontuação fixa opta mais por mismatches do que o sistema de pontuação variável; mesmo a pontuação para mismatches no sistema de pontuação fixa sendo geralmente menos recompensadora.

O sistema de pontuação variável usa mais inserções e deleções, quando conveniente, visando o alcance de good mismatches (que resultam numa boa pontuação). A escolha de inserções e deleções nos alinhamentos com sistema de pontuação variável se refletiu no tamanho dos alinhamentos: $\alpha_{\rho\tau_1 SPV}$, $\alpha_{\rho\tau_2 SPV}$ e $\alpha_{\rho\tau_5 SPV}$ foram maiores respectivamente do que $\alpha_{\rho\tau_1 SPF}$, $\alpha_{\rho\tau_2 SPF}$ e $\alpha_{\rho\tau_5 SPF}$.

A análise qualitativa revelou algumas ocorrências, como ilustrado pela Figura 5.15. O alinhamento das linhas 1 a 3 foi realizado com o sistema de pontuação variável; já, o alinhamento das linhas 4 a 6 utilizou o sistema de pontuação fixa. A sequência das linhas 1 e 4 correspondem ao padrão-ouro e nas linhas 3 e 6 está o mesmo traço faltoso.

A área oval em azul no item (a) da Figura 5.15 (marcada com o símbolo ↖) mostra a subsequência 1430 (eventos `RcvInvoke` e `Tr-invoke.ind`, respectivamente) na linha 1, colunas 11-12 alinhada a outra subsequência 1430 (match) na linha 3. A área oval em vermelho (marcada com o símbolo ↑) mostra que o evento 14 na linha 4, coluna 10 foi alinhado a espaços. O alinhamento de um evento do padrão-ouro a espaços no traço significa que o evento não foi realizado quando o traço foi executado. Porém, vendo a área em oval em azul sabemos que este evento também foi realizado no traço. O sistema de pontuação fixa não alinhou corretamente os eventos. Outra situação errônea ocorreu com o evento seguinte. O evento 30 foi incorretamente alinhado ao evento 20 como se houvesse uma mutação no traço.

O alinhamento incorreto dos eventos 14 e 30 pelo sistema de pontuação fixa causou ainda outros alinhamentos incorretos em virtude deste primeiro erro como mostram as áreas destacadas 1 e 2 no item (b) da Figura 5.15.

A Figura 5.16 mostra as diferenças em alguns trechos alinhados com os dois sistemas de pontuação. O SPV_{WTP} foi utilizado no primeiro alinhamento (mais longo). O alinhamento mais curto foi obtido com o SPF_{WTP}. A primeira sequência, nos dois alinhamentos, é o padrão-ouro

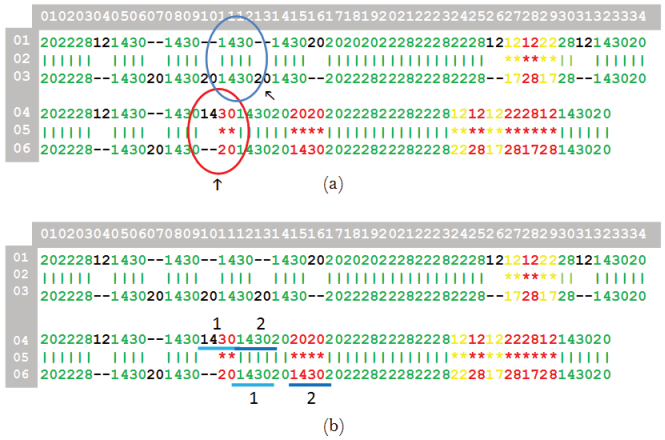


Figura 5.15: Avaliação qualitativa dos alinhamentos com SPV e SPF no WTP

e a segunda, é o traço de execução coletado no teste de robustez. As regiões 1, 2 e 3 obtidas com SPF_{WTP} correspondem, respectivamente, às regiões A, B e C obtidas com o SPF_{WTP}.

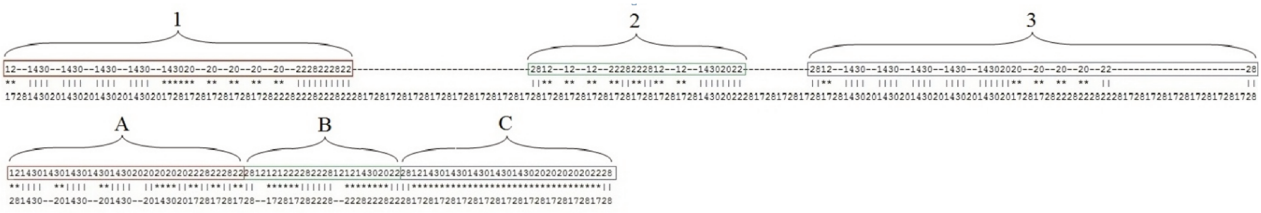


Figura 5.16: Diferença no tamanho dos alinhamentos com SPV_{WTP} e SPF_{WTP} no WTP

O fato do alinhamento resultante da adoção do sistema de pontuação fixa ser geralmente menor do que o alinhamento que adota o sistema de pontuação variável mostra que o fato de o SPF_{WTP} não ter informações sobre a similaridade dos pares de forma individual faz com que despreze algumas possibilidades de alinhamento importantes. Algumas vezes pudemos perceber, neste estudo de caso, que alinhamentos de eventos muitos similares (embora diferentes) foram desprezados. O fato de trabalharmos com sistemas que possuem eventos de entrada e saída justifica o uso de um sistema de pontuação que sabe pontuar melhor pares entrada-entrada, saída-saída do que pares entrada-saída ou saída-entrada.

Não somente os números das Tabelas 5.6 e 5.7 apresentam resultados relevantes para a classificação dos traços. O alinhamento em si é uma fonte de informação para o testador, principalmente em casos em que a similaridade de um traço se aproxima do limiar definido ou quando não há limiar, como no caso do estudo de caso do WTP aqui apresentado.

Tomando como exemplo o traço τ_4, que tem o alinhamento com SPV_{WTP} ilustrado na Figura 5.17, podemos visualizar claramente o trecho onde ocorreu a não-robustez. Para entendermos melhor os resultados obtidos vamos explicar o comportamento representado por τ_4.

Quando o Responder envia um resultado ao Initiator, ele inicia um timer enquanto espera pela resposta. De acordo com a especificação do WTP, quando o timer expira, o Responder recebe um evento `TimerTo_R` (evento 17), retransmite a mensagem de resultado (evento 28) e inicia novamente o timer para esperar pela resposta. Este procedimento é repetido um número definido de vezes, após as quais a transação é abortada e uma indicação é enviada à camada superior.

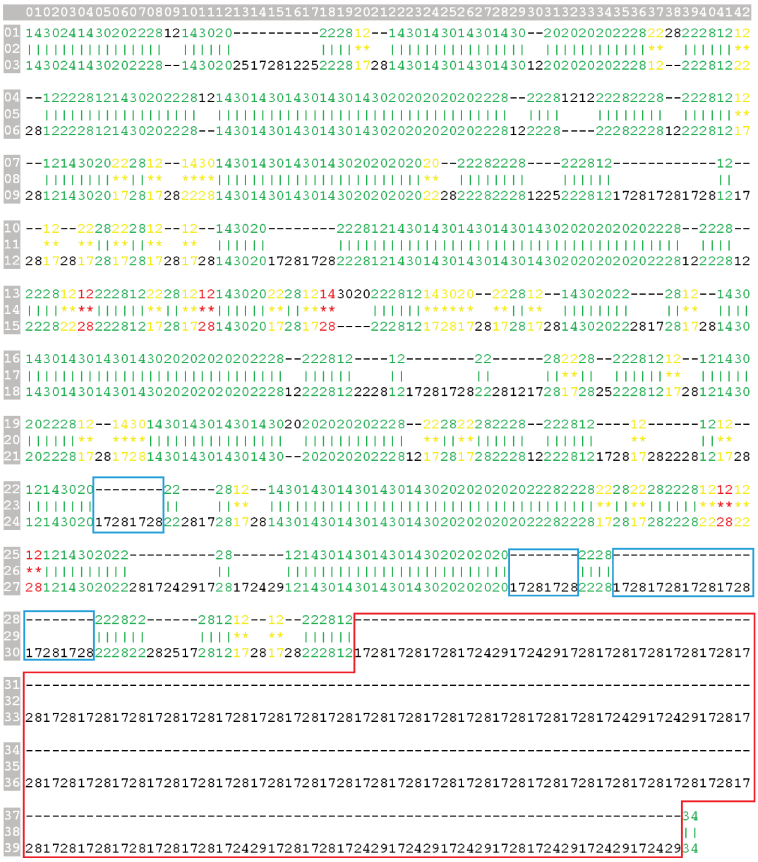


Figura 5.17: Alinhamento do traço τ_4 do WTP com SPV_{WTP}

Na Figura 5.17, os retângulos menores nas linhas 22-24, 25-27 e 28-30 destacam comportamentos robustos. Tais comportamentos foram considerados como sendo robustos porque o número de vezes que o timer foi disparado estava dentro dos limites estabelecidos na especificação do SUT. Entretanto, olhando para a área selecionada na parte final do alinhamento (linhas 28-39), vemos que houve muitas inserções de eventos, ou seja, ocorreram muitos eventos no WTP que não estavam previstos no padrão-ouro. O disparo de inúmeros timers ocorreu pois o sistema entrou em livelock, ou seja, o número de timers disparados não foi limitado quando chegou ao limite definido na especificação do protocolo.

Concluimos que, quando as sequências são muito pequenas, o sistema de pontuação mais sofisticado não gera melhores resultados do que o sistema de pontuação mais simples. Porém,

sabemos que a maioria dos sistemas não gera traços com poucos eventos. Em geral, os traços apresentam centenas ou milhares de eventos, o que dificulta a avaliação e justifica a escolha do sistema de pontuação que seja capaz de gerar alinhamentos mais significativos.

Em termos de classificação dos traços em robustos e não robustos, as curvas ROC mostram, através da medida da área sob a curva, que tanto a pontuação fixa quanto a variável são boas escolhas. No entanto, para auxílio ao diagnóstico, o sistema de pontuação variável é mais adequado, pois gera alinhamentos semanticamente mais próximos do esperado. Isto nos leva a concluir que a GCSpec deve utilizar o sistema de pontuação variável, confirmando Q2.

5.5 Uso do Algoritmo de Alinhamento Semi-Global

A maioria dos sistemas possui eventos que marcam o início e o final da execução de um caso de teste. O WTP, por exemplo, utiliza o evento `RcvInvoke` para identificar o começo de cada nova transação. Durante os testes de robustez, algumas vezes não é possível coletar os traços de maneira total, ou seja, trechos de traços que contém os eventos utilizados como marcadores não são coletados pois o comportamento do SUT é armazenado apenas durante um intervalo de tempo que engloba o período em que a falha é injetada.

A Figura 5.18 ilustra o que pode ocorrer durante a coleta dos traços para alguns SUT. Nesta figura podemos perceber que apenas o traço (c) contém os eventos que correspondem ao início e final do caso de teste. Assim, este traço é mais adequado ao algoritmo de alinhamento global. Os demais traços também podem ser alinhados utilizando-se o alinhamento global mas, nos casos (a), (b) e (d), o mais adequado é o uso do algoritmo de alinhamento semi-global.

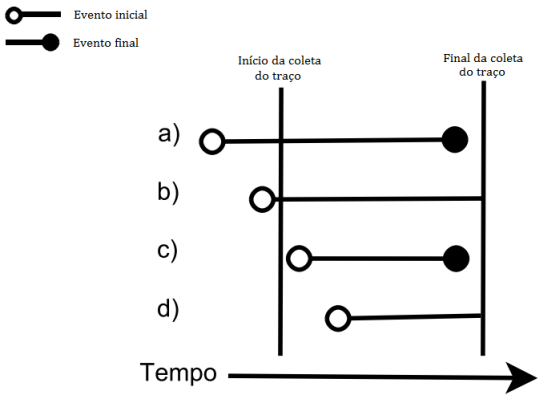


Figura 5.18: Situações que podem ocorrer durante a coleta dos traços (adaptado de [13])

O algoritmo semi-global, descrito na Seção 3.4, não penaliza inserções e/ou deleções de espaços nas extremidades do alinhamento. Assim, em caso de uma das sequências ser como nos exemplos (a), (b) ou (d), o algoritmo é capaz de alinhar as partes coincidentes da execução do caso de teste e desprezar as partes excedentes no início ou final das sequências. A não inclusão de

prefixos ou sufixos alinhados a espaços melhora a pontuação total do alinhamento pois permite que as penalidades atribuídas a estas áreas sejam desprezadas.

5.6 Trabalhos Relacionados

Há diferentes abordagens utilizadas atualmente para análise de resultados de teste de robustez [56]. A nossa proposta utiliza uma nova solução para reduzir o número de falso-positivos que envolve o uso de algoritmos de alinhamento de sequências de forma a estender a comparação com o padrão-ouro.

Além da nossa abordagem, existem poucos trabalhos que utilizam algoritmos de alinhamento de sequências fora do contexto da bioinformática. Dentre eles, podemos citar como exemplo o trabalho de Bose e colegas que utiliza algoritmos de alinhamento múltiplo de sequências em mineração de processos (*process mining*), técnica que tenta extrair informações sobre os processos a partir de traços de execução [15]. Eles também utilizam os alinhamentos como ferramenta de avaliação visual. Porém, diferentemente de nossa abordagem, eles não assumem a existência de um modelo do sistema para guiar o alinhamento.

No trabalho de Coll e Szymanski [26], os autores utilizam alinhamento semi-global de pares de sequências para descobrir sinais de intrusão em dados de tráfego de rede. Eles analisaram um tipo de ataque em que o intruso tenta se passar por um usuário legítimo para ganhar acesso ao sistema. Os autores criaram sequências de eventos que são padrões de comportamento de usuários legítimos e comparam tais padrões aos traços coletados da rede durante sua operação normal. Eles testaram três diferentes sistemas de pontuação. Um dos sistemas de pontuação utilizava pontuações fixas similarmente ao SPF utilizado em nossos estudos de caso. Os dois outros sistemas de pontuação testados foram baseados no primeiro porém agrupam (com uma pontuação única) os eventos de acordo com sua similaridade. O modo de definição dos agrupamentos e a pontuação utilizada para cada agrupamento de eventos foram baseados na pontuação fixa. Diferentemente de nossa proposta de definição do sistema de pontuação variável (apresentada no Capítulo 4), os autores não citam se foi definida uma abordagem para a obtenção dos valores aplicados nos sistemas de pontuação.

Al-Ibaisi e outros autores [1] também analisaram tráfego de rede para verificar diferentes tipos de ataques através do alinhamento global de sequências. Eles não detalharam o sistema de pontuação utilizado mas mencionaram que o alinhamento entre um modelo de comportamento e um comportamento observado é classificado segundo um limiar pré-definido. Os alinhamentos com pontuação abaixo deste limiar foram considerados como anômalos.

Uyar et al [96] usam alinhamento de pares de sequências para avaliação de fitness (taxa definida em função do número de matches dividido pelo tamanho das sequências) em casos de teste gerados através de um método meta-heurístico. Após gerarem os casos de teste, os autores avaliam o fitness de cada caso de teste em relação a um padrão pré-estabelecido. Este padrão contém um conjunto de eventos que todo bom caso de teste deve conter. O objetivo é verificar se o método meta-heurístico é capaz de gerar casos de teste apropriados.

Kessentini et al [52] desenvolveram o trabalho que mais se assemelha à nossa proposta. Eles também desenvolveram um oráculo, porém o uso é para verificar o resultado de testes de transformação de modelos de referência. Os autores citam que têm um grande número de transformações realizadas em modelos e, após rodar cada uma das transformações, utilizam o alinhamento de sequências para avaliar a validade dos modelos transformados. Eles adotaram o algoritmos de alinhamento global para determinar quão similares são as transformações.

5.7 Considerações Finais

A avaliação de resultados é, para todas as técnicas de teste utilizadas neste capítulo, dependente do SUT. Assim, como dissemos na Seção 2.3 do Capítulo 2, é impossível construir um oráculo genérico. Desta maneira, alguns passos da GCSpec consistem da especialização da abordagem para cada SUT para o qual a abordagem é utilizada.

Os oráculos comumente utilizados para avaliação de resultados de teste de robustez [83, 58, 19] realizam comparação exata o que não é adequado para este tipo de teste porque mesmo o SUT robusto pode vir a ter um comportamento diferente do padrão-ouro, devido ao disparo de mecanismos de tolerância a falhas. Nossa abordagem permite a comparação de traços obtidos na presença de falhas com padrões de comportamento, considerando o mesmo workload, pois os algoritmos de alinhamento de pares de sequências são algoritmos de matching inexato.

Definimos o conceito da métrica de similaridade obtido a partir da pontuação do alinhamento que pode, aliada a um limiar pré-definido, ser utilizada na classificação dos traços do SUT. O alinhamento das sequência pode, ainda, ser uma ferramenta de avaliação visual auxiliando o testador na localização dos erros e no entendimento do contexto em que os mesmos ocorreram, complementando o veredicto apresentado.

O algoritmo de alinhamento de sequências não precisa necessariamente receber como entradas as sequências iniciadas no mesmo ponto de execução do SUT. Isto configura uma vantagem em relação à abordagens baseadas em mineração de processos (*process mining*) [15].

A abordagem apresenta como desvantagem em relação à comparação com o padrão-ouro tradicional a necessidade de haver uma especificação ou modelo do SUT para construção da árvore de categorização de eventos. O testador precisa conhecer os eventos do alfabeto do SUT, de forma a poder definir, na árvore de categorização de eventos, como eles se relacionam.

A razão para termos desenvolvido um oráculo baseado na especificação é o fato de verificarmos que o oráculo tradicional de comparação com o padrão-ouro não é totalmente confiável pois leva a uma alta taxa de falsos-positivos. Além disso, este tipo de oráculo baseia o veredicto de robustez apenas no resultado colhido a partir do próprio (que pode apresentar falhas de implementação). Em nossa abordagem, o padrão-ouro continua sendo uma referência, mas a execução durante o teste de robustez não necessariamente precisa ser idêntica à este modelo de comportamento.

Nossa abordagem ainda pode ser empregada sem o uso da especificação (com o sistema de pontuação fixa). Os resultados, em termos da qualidade do alinhamento, podem não ser tão acurados pelo fato deste sistema de pontuação não conhecer o relacionamento entre os eventos.

Isto pode levar a situações como o alinhamento de eventos de entrada à eventos de saída mas é um avanço em relação à abordagem tradicional de comparação com o padrão-ouro pois permite o matching inexato.

Capítulo 6

Aplicação do Algoritmo de Alinhamento Local

Este capítulo apresenta a segunda abordagem desenvolvida neste trabalho. Esta abordagem, chamada de PSSpec (Property Search based on the Specification), assim como a GCSpec, também é utilizada para análise de resultados de teste de robustez. Porém, diferentemente da abordagem apresentada no capítulo anterior, a PSSpec é baseada na busca de propriedades de segurança (*safety*) em traços de execução coletados durante o teste. Para a comparação de cada traço de execução do SUT com uma propriedade pré-estabelecida usamos o algoritmo de alinhamento local de pares de sequências, descrito na Seção 3.5. A abordagem PSSpec consiste da evolução da tradicional abordagem de "busca por propriedades" descrita na Seção 2.3.

A Seção 6.1 posiciona a PSSPec no contexto da busca por propriedades e detalha suas origens. A visão geral consta da Seção 6.2. O funcionamento da abordagem é descrito na Seção 6.3. Na Seção 6.4 descrevemos o estudo de caso realizado para prova de conceito da abordagem proposta. A seguir, na Seção 6.5 estão os trabalhos relacionados. Por fim, a Seção 6.6 contém as considerações finais deste capítulo.

6.1 Introdução

Como dissemos no Capítulo 2, a busca por propriedades é uma abordagem bastante utilizada não somente como oráculo de teste mas também em outras áreas como, por exemplo, na detecção de defeitos [65]. Já, o trabalho de Cavalli e colegas [19], utiliza a busca por propriedades, chamadas pelos autores de invariantes, para a avaliação de resultados de injeção de falhas. No trabalho citado, a busca de cada propriedade em traços de execução é realizada com o uso de algoritmos clássicos de busca de padrões [16, 53].

Em nossa abordagem mantivemos a busca por propriedades comparando-as com os traços de execução, similarmente ao que é feito em [19]. Entretanto, o mecanismo que realiza as buscas foi substituído pelo uso de algoritmos de alinhamento de pares de sequências. Escolhemos,

neste caso, o uso do alinhamento local pois as sequências envolvidas (propriedade e traço de execução) apresentam tamanhos muito diferentes [92]. O traço geralmente é muito maior do que a propriedade a ser buscada, o que indica o uso do algoritmo de alinhamento local, utilizado para o alinhamento de regiões com alta similaridade.

Em um contexto geral, propriedades de segurança (*safety*) declaram que algo ruim não acontece durante a execução do sistema [55] e podem ser buscadas através da declaração de sua negativa. Em nosso trabalho, cada propriedade é buscada através do cenário de não-robustez que pode afetar o SUT, ou seja, um "mau comportamento" que caracteriza uma situação em que há falha de segurança (*safety*). Por exemplo, a propriedade de segurança "o carro deve desligar o controle de cruzeiro quando o motorista acelerar o veículo" é buscada como cenário de não-robustez através de sua negativa "o carro não desliga o controle de cruzeiro quando o motorista acelera o veículo". Assim, no contexto da PSSpec, quando citamos propriedades de segurança nos referimos à *safety*.

As propriedades utilizadas para busca nos traços podem ser obtidas através de técnicas de análise de segurança (*safety*), como as árvores de falhas [94]. Tais árvores mostram diferentes cenários que conduzem à falhas catastróficas ou através da própria especificação do SUT.

6.2 Visão Geral da Abordagem

Nesta seção temos a visão geral da PSSpec. Seus passos são descritos a seguir:

1. Definição do conjunto de propriedades de segurança $\varrho = \{\varrho_1, \varrho_2, \ldots, \varrho_t\}$ de tamanho t, sendo $t \in \mathbb{N}$ o número de propriedades definidas pelo testador;

2. Coleta dos traços de execução que formam o conjunto $\tau = \{\tau_{11}, \tau_{12}, \ldots, \tau_{rs}\}$, de tamanho $|r * s|$, onde r é o número de casos de teste e s é o número de falhas injetadas durante o teste de robustez;

3. Filtragem e codificação dos traços τ e das propriedades ϱ para a manutenção apenas das informações relevantes ao alinhamento. Na PSSpec, diferentemente da GCSpec, utilizamos tanto informações do fluxo de controle quanto informações sobre parâmetros e seus valores durante a comparação;

4. Definição da função de pontuação (ω), conforme o método descrito na Seção 4.2;

5. Alinhamento de todas as propriedades do conjunto ϱ a cada cada traço $\tau_{ij} \in \tau$ com o algoritmo de alinhamento local de pares de sequências;

6. Obtenção das saídas de cada alinhamento:

   - o alinhamento ótimo entre ϱ_k e τ_{ij} denotado por $\alpha_{ot(\omega)}(\varrho_k, \tau_{ij})$ com relação a um sistema de pontuação ω, onde $1 \leq k \leq t$, $1 \leq i \leq r$ e $1 \leq j \leq s$;
   - a pontuação do alinhamento ótimo ($\wp\alpha_{ot(\omega)}(\varrho_k, \tau_{ij})$);

- a métrica de similaridade ($\mu_{\omega}(\varrho_k, \tau_{ij})$).

Ao contrário do que ocorre na GCSpec, na PSSpec quanto mais similar o traço τ_{ij} é em relação à propriedade ϱ_k maior a chance de que ele seja não-robusto. Caso a propriedade ϱ_k não seja localizada em τ_{ij}, ou seja, caso a pontuação do alinhamento $\wp_{\alpha\omega}(\varrho_k, \tau_{ij})$ seja muito baixa, não podemos afirmar que o traço é robusto; apenas podemos dizer que ele não possui o comportamento de não-robustez descrito por ϱ_k. Assim, podemos apenas afirmar que, para o caso de teste i, o SUT não apresentou o comportamento não-robusto k, mesmo durante a injeção da falha j.

6.3 Detalhamento dos Passos

As propriedades que compõem o conjunto ϱ representam cenários de não-robutez e são utilizadas para verificação dos traços do conjunto τ. Como dissemos anteriormente, elas podem ser obtidas através de análise de segurança (*safety*) ou através da espcificação do SUT.

Neste trabalho, cada propriedade a ser buscada é uma sequência de eventos pertencentes ao alfabeto Σ representada por um cenário definido como $\varrho =< e_1, e_2, e_3, \ldots, e_n >$ em que cada item e_i, $1 \leq i \leq n$ representa um evento. Cada evento pode ter parâmetros associados a ele. Cada parâmetro definido deve apresentar um valor associado. O valor de cada parâmetro pode ser desde um valor simples (por exemplo, "Falso") até um conjunto de valores (por exemplo, o intervalo $[0, 100]$). O valor de cada parâmetro representa uma partição do domínio de entrada do parâmetro ao qual ele está associado.

A Figura 6.1 mostra um exemplo de uma propriedade de segurança, o cenário de não-robustez derivado da propriedade e a sequência de eventos obtida a partir do cenário.

Propriedade: o sistema de controle de cruzeiro é **desligado** após o motorista **acelerar** o veículo

Cenário de não-robustez: o sistema de controle de cruzeiro NÃO é **desligado** após o motorista **acelerar** o veículo

Sequência: recordSpeed, enableControl, accelerate, getSpeed, off

Sequência codificada: 33, 29, 15, 37, 17

Figura 6.1: Exemplo de propriedade mapeada para uma sequência de eventos a ser procurada em traços de execução

A Tabela 6.1 mostra os parâmetros associados aos eventos da sequência definida na Figura 6.1 e os valores que cada parâmetro pode receber.

O passo seguinte da abordagem é o pré-processamento dos traços de execução para manter apenas as informações relevantes ao alinhamento. Na GCSpec, são eliminadas todas as informações exceto o fluxo de controle do SUT pois não levamos em conta os parâmetros durante a comparação com o padrão-ouro. Já, na PSSpec, permanecem o fluxo de controle, os parâmetros e seus valores. Assim, a codificação na PSSpec é ligeiramente diferente do que ocorre na GCSpec onde cada evento recebe uma codificação única.

Tabela 6.1: Parâmetros e valores relacionados aos eventos da propriedade da Figura 6.1

Evento	Parâmetro	Valor Selecionado	Valor Desconsiderado
recordSpeed	speed	70	< 70 ou > 70
enableControl	-	-	-
accelerate	speed	71 a 75	< 71 ou > 75
getSpeed	-	-	-
off	-	-	-

Na PSSpec cada evento pode receber diferentes valores de codificação: um código é atribuído ao evento no traço quando ele está acompanhado do parâmetro com o valor selecionado e outros códigos são atribuídos às demais partições que contém os demais valores possíveis no domínio de entrada do parâmentro.

Para possibilitar que a codificação levasse em conta os parâmetros e seus valores, redefinimos a função de codificação utilizada pela GCSpec (descrita na Equação 5.1). Na PSSpec, utilizamos a função de codificação conforme a Equação 6.1. Seja $\mathcal{D}$ o conjunto que representa o evento $\sigma \in \Sigma$, acompanhado (quando necessário) de parâmetros $p_i \in P_r$ e dos valores selecionados v_i. Então temos função de codificação para a PSSpec definida conforme a Equação 6.1:

$$C_{\mathcal{D}} : \mathcal{D} \rightarrow \mathbb{N} \tag{6.1}$$

Esta diferença na codificação é representada com a criação de um novo nível na árvore de categorização de eventos usada para composição da matriz de pontuação. A matriz é elaborada conforme o método descrito no Capítulo 4. Porém, um novo nível é definido abaixo dos que representam os eventos do SUT (nós-filho de Σ_{ev}) para representar as codificações possíveis de cada evento (de acordo com as partições definidas, conforme exemplo da Tabela 6.1). A Figura 6.2 ilustra a criação de um novo nível para alguns nós na árvore.

O aumento de um nível na árvore gera modificações também nas matrizes de distância D e de pontuação $\mathcal{S}$ que precisam ser estendidas para a inclusão das distâncias e pontuações relativas a estes novos nós. Assim, cada novo nó da árvore da oridem a uma nova linha e uma nova coluna nas matrizes D e $\mathcal{S}$.

Experimentos preliminares que realizamos com o algoritmo de alinhamento de sequências mostraram que o número de falso-negativos para o alinhamento de traços à propriedades é bastante significativo. Isto implica que o traço muitas vezes contém o cenário de não robustez descrito na propriedade mas este não é encontrado.

Analisando os resultados verificamos tais ocorrências foram causadas pela característica do alinhamento local de permitir apenas alinhamentos com pontuação positiva. Como a propriedade é, geralmente, muito menor do que o traço ocorrem muitas inserções de eventos. Assim, a penalização pela ocorrência de espaços acabava muitas vezes por mascarar a exisstência de bons alinhamentos.

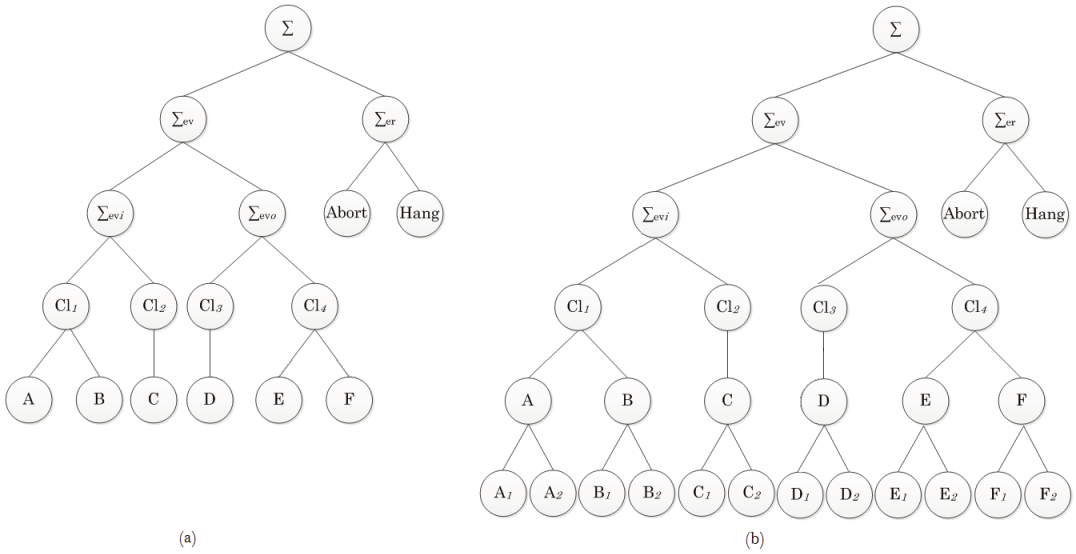


Figura 6.2: (a) Árvore de categorização de eventos (b) Novo nível inserido na árvore

Resolvemos então, no estudo de caso final deste trabalho, utilizar não somente o algoritmo tradicional de alinhamento local de sequências proposto por Smith e Waterman (que utiliza o mesmo valor para todo indel no alinhamento) mas também outro algoritmo que utiliza a função de penalização atenuada de espaços descrito em [51].

Como descrito na Seção 3.5, o algoritmo de alinhamento local também constrói a matriz de similaridades (M_s) de acordo com as Equações 3.5, 3.6 e 3.7. Com a utilização da função de penalização atenuada, passamos a utilizar outro algoritmo que constrói M_s a partir de três matrizes auxiliares de mesma ordem, onde cada combinação de pontuação para espaços é calculada.

Nesta implementação do algoritmo que trabalha com a penalização atenuada para espaços, a complexidade de tempo permanece idêntica à complexidade do algoritmo de alinhamento com função de penalização constante, porém a complexidade de espaço é multiplicada pelo número de matrizes auxiliares que o algoritmo utiliza para o cálculo da melhor opção de composição do alinhamento. Isto não chega a ser preocupante no caso da PSSpec pois as propriedades que definimos têm tamanho próximo a uma constante. Assim, o espaço que cada uma das matrizes ocupa é aproximadamente linear.

Após o cálculo de M_s, temos as seguintes saídas da abordagem PSSpec:

- As sequências alinhadas de forma "ótima" ($\alpha_{ot(\omega)}(\varrho_k, \tau_{ij})$);
- A pontuação do alinhamento ótimo ($\wp_{\alpha_{ot(\omega)}}(\varrho_k, \tau_{ij})$);
- O valor da métrica de similaridade ($\mu_\omega(\varrho_k, \tau_{ij})$).

Como o alinhamento local tem sempre pontuação mínima igual a zero (conforme explicado na Seção 3.5) a métrica de similaridade para a PSSpec é calculada sem a necessidade de considerarmos alinhamentos com pontuação negativa, diferentemente do que fizemos na GCSpec.

Assim, a métrica de similaridade é calculada de forma mais simples, conforme ilustra a Equação 6.2.

$$\mu_{\omega}(\varrho_k, \tau_{ij}) = \frac{\wp_{\alpha_{ot(\omega)}}(\varrho_k, \tau_{ij})}{max(\wp_{\alpha_{\omega}}(\varrho_k, \tau_{ij}))} \tag{6.2}$$

em que $max(\wp)$ é a pontuação máxima possível entre ϱ_k e τ_{ij} (situação ilustrada pela Figura 6.3).

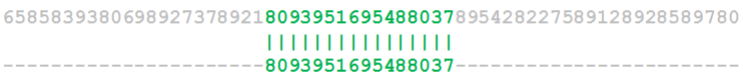


Figura 6.3: Exemplo de alinhamento ótimo entre um traço de execução e uma propriedade sem espaços entre os eventos procurados

6.4 Estudo de Caso

Esta seção descreve o estudo de caso realizado com o intuito de exemplificar o uso da abordagem PSSpec como extensão da técnica de oráculo baseado na busca de propriedades. Utilizamos como estudo de caso na PSSpec o sistema Elevator tanto para análise quantitativa quanto qualitativa. Realizamos o estudo quantitativo dividindo o processo nas seguintes fases: definição, planejamento, execução, sumarização dos resultados e conclusão.

6.4.1 Definição

Este experimento tem como propósito verificar o desempenho dos algoritmos de alinhamento local com dois diferentes modos de penalizar os espaços nos alinhamentos durante a busca por propriedades em traços de execução:

- algoritmo de Smith-Waterman com penalização constante para cada espaços adicionado ao alinhamento;
- algoritmo de alinhamento local com penalização atenuada de espaços, conforme implementação proposta por Altschul et al [51].

Objetivo: Verificar no alinhamento local de sequências de teste de robustez, se a penalização atenuada de espaços leva a melhores resultados quando comparada com a penalização constante.

- **Q:** A penalização atenuada utilizada no sistema de pontuação variável pode resultar em melhores alinhamentos quando procuramos cenários de não-robustez em traços de execução provenientes da injeção de falhas quando comparados ao uso da penalização constante?

- **M:** A similaridade do alinhamento obtido quando adotamos a penalização de espaços atenuada juntamente ao sistema de pontuação variável baseado na especificação do SUT é maior do que a quando adotamos a penalização constante dos espaços.

6.4.2 Planejamento

Os dados de entrada e saída deste experimento são os seguintes:

- Variáveis independentes (dados de entrada):
  - Sequências coletadas a partir da execução do SUT sem injeção de falhas (padrões-ouro).
  - Sequências coletadas a partir da injeção de falhas (traços resultantes do teste de robustez).
  - Sistemas de pontuação.
- Variável dependente (dado calculado):
  - Similaridade: valor obtido pela métrica de similaridade para os traços avaliados.

Para este experimento definimos quatro propriedades de segurança (safety) que formam o conjunto $\varrho = \{P1, P2, P3, P4\}$. Derivamos, para cada uma delas um cenário de não-robustez e, então, a sequência a ser buscada nos traços do SUT.

Propriedade 1 (P1): Elevador não se move enquanto a porta está aberta. Cenário de não-robustez: Elevador se move antes de fechar a porta.

```
P1=<requestUp,addStop,motorMoveUp,motorStop,closeDoor,openDoor,requestUpServiced>
```

P1 após codificação: 43, 46, 50, 52, 48, 47, 82

Propriedade 2 (P2): Elevador não abre a porta enquanto está em movimento. Situação de não robustez: Elevador abre a porta antes de parar no andar selecionado.

```
P2=<requestUp,addStop,motorMoveUp,closeDoor,openDoor,motorStop,requestUpServiced>
```

P2 após codificação: 43, 46, 50, 48, 47, 52, 82

Propriedade 3 (P3): Elevador abre a porta quando para no andar selecionado. Situação de não robustez: Elevador para e não abre a porta.

```
P3=<requestUp,addStop,closeDoor,motorMoveUp,motorStop,requestUpServiced>
```

P3 após codificação: 43, 46, 50, 48, 52, 82

Propriedade 4 (P4): Elevador deve iniciar cumprimento de requisição com o fechamento da porta e o movimento para o andar selecionado. Situação de não robustez: Elevador fecha a porta e depois a abre novamente. Somente após fechar a porta pela segunda vez segue para o andar selecionado.

```
P4=<requestUp,addStop,motorMoveUp,closeDoor,openDoor,closeDoor,motorStop,
                        openDoor,requestUpServiced>
```

P4 após codificação: 43, 46,50, 48, 47, 48, 52, 47, 82

Para todas as propriedades do conjunto ϱ, os parâmetros e valores selecionados para busca na sequência são os mesmos, conforme descrito na Tabela 6.2.

Tabela 6.2: Parâmetros e valores relacionados aos eventos das propriedades de ϱ

Evento	Parâmetro	Valor Selecionado	Valor Desconsiderado
requestUp, addStop	floor	1	$\neq 1$
motorMoveUp, closeDoor, openDoor,	elevator	1	$\neq 1$
requestUpServiced	-	-	-

Os traços utilizados neste experimento compõem o conjunto τ_{PSSpec}. O conjunto τ_{PSSpec} foi obtido a partir de τ, o conjunto de traços utilizados no experimento do sistema Elevator para validação da GCSpec. Assim, $\tau_{PSSpec} \subset \tau$, sendo que $|\tau| = 3200$ e $|\tau_{PSSppec}| = 810$.

Todos os traços de τ foram gerados durante o teste de robustez por meio da injeção estática de falhas. Após avaliação manual dos traços, selecionamos o conjunto τ_{PSSpec}. Todos os traços deste conjunto tem em comum o fato de não satisfazerem as propriedades em ϱ.

Então, a partir de τ_{PSSpec}, para cada uma das propriedade de ϱ, criamos outro conjunto τ_{PSSpec_i}, com $1 \leq i \leq 4$ utilizando estratégias de geração de traços errôneos similares às citadas em [13, 65]:

- τ_{PSSpec_1}: 405 traços tiveram o evento relacionado ao ato de movimentar o elevador (evento 50) invertido em relação ao evento de fechamento da porta (evento 48). Os outros 405 traços permaneceram inalterados;

- τ_{PSSpec_2}: 410 traços tiveram a posição do evento que representa o ato de abrir a porta do elevador (evento 47) alterada para que o evento ficasse posicionado antes dos eventos de parada do elevador (evento 52). Os outros 400 traços não foram alterados;

- τ_{PSSpec_3}: 410 traços tiveram o evento relacionado ao ato de abrir a porta do elevador omitido (evento 47). Os demais 400 eventos do conjunto não foram alterados;

- τ_{PSSpec_4}: 405 traços tiveram dois eventos (relacionados ao fechamento e à abertura da porta - respectivamente 48 e 47) duplicados. Os 405 eventos restantes permaneceram inalterados.

A justificativa para a geração manual de traços com erros conhecidos foi a necessidade de construirmos um experimento controlado para possibilitar a avaliação do algoritmo de alinhamento local. Assim, pudemos avaliar o alinhamento entre as propriedades de ϱ e os traços provenientes da injeção de falhas. Esta estratégia nos permitiu conhecer antecipadamente cada um dos traços que continha os eventos nas posições originais e quais traços haviam sido alterados facilitando a análise de resultados.

Por questão de espaço não foi possível apresentar nesta tese todos os resultados obtidos com os estudos de caso. O registro completo pode ser encontrado em nossa página[1].

6.4.3 Execução

Como descrito na Seção 6.3, na PSSpec, o sistema de pontuação é formado por uma matriz de substituição (S) e por uma função de penalização de espaços, assim como na GCSpec. Na PSSpec, a composição da matriz S é também baseada na árvore de categorização de eventos. Porém, a árvore contém um nível abaixo dos nós que representam os eventos. Este nível especializa a descrição dos eventos de forma a categorizá-los de acordo com os parâmetros que os acompanham e seus respectivos valores.

A Figura 6.4 ilustra um trecho da árvore de categorização de eventos do sistema Elevator. Neste trecho podemos verificar a existência de um novo nível (com os eventos destacados em negrito), em que mostramos o nó que representa o evento `requestDown` expandido de forma a representar a ocorrência do evento associado aos diferentes valores que o parâmetro pode assumir.

6.4.4 Sumarização dos Resultados e Conclusões

Para o estudo de caso Elevator alinhamos todos os traços de cada um dos conjuntos τ_{PSSpec_i}, com $1 \leq i \leq 4$ às propriedades do conjunto $\varrho = \{P1, P2, P3, P4\}$ utilizando a matriz de pontuação construída com base na especificação e as penalização constante e atenuada para espaço. Além disso, classificamos todos os traços manualmente.

Na PSSpec não utilizamos limiares pois não temos um conjunto de padrões-ouro a partir do qual possamos obter um valor adequado a esta variável. Assim, os traços são avaliados após serem classificados de acordo com o grau de similaridade em relação a cada uma das propriedades. A classificação dos traços existentes cria uma ordenação ("ranking") onde o traço mais similar à propriedade apresenta menor chance de ser robusto. Vale salientar que mesmo que o traço não apresente grande similaridade em relação a uma determinada propriedade não podemos afirmar que ele é robusto.

[1] www.ic.unicamp.br/~glemos/Resultados/ResultadosPSSpecElevator.pdf

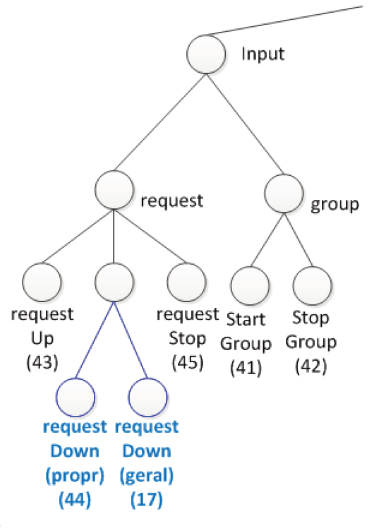


Figura 6.4: Parte da árvore de categorização de eventos do sistema Elevator

A inexistência de um limiar nos levou a novamente escolher a curva ROC para a avaliação da PSSpec em relação aos modos de penalização de espaços. A Figura 6.5 mostra as curvas ROC para cada uma das propriedades considerando-se a penalização constante de espaços e a penalização atenuada de espaços .

Calculamos também a área sob as curvas ROC de modo a comparar a qualidade dos sistemas de pontuação (com penalização constante de espaços e com penalização atenuada de espaços). A Tabela 6.3 contém os resultados obtidos.

Tabela 6.3: Áreas sob as curvas ROC obtidas nos experimentos realizados com a PSSpec

Propriedade	Área Curva ROC penalização Constante	Área Curva ROC penalização Atenuada
P1	0,992	0,996
P2	0,998	1,000
P3	0,887	0,985
P4	1,000	0,999

O algoritmo de alinhamento local mostrou-se, através dos experimentos conduzidos com o sistema Elevator, adequado ao uso como oráculo de teste. Com os resultados obtidos através da análise das áreas sob as curvas pudemos confirmar que em três das quatro propriedades buscadas a penalização atenuada saiu-se melhor do que a penalização constante.

Além dos resultados quantitativos, fizemos verificações em termos qualitativos que nos ajudaram a compreender as vantagens da PSSpec em relação às abordagens de busca de padrões existentes. Uma das vantagens é o fato de que, através do sistema de pontuação, montado com a árvore de categorização de eventos, o algoritmo é capaz de saber quais eventos são semanti-

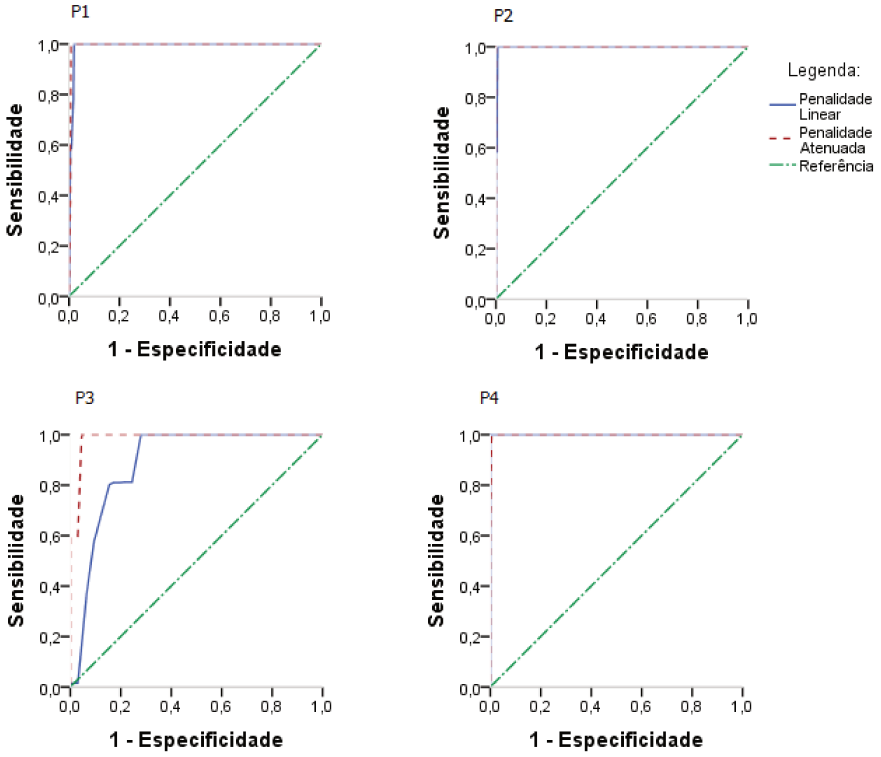


Figura 6.5: Curvas ROC obtidas para cada uma das propriedades analisadas

camente similares aos eventos buscados (e alinhá-los ao traço), mesmo que estes não estejam explicitamente descritos na propriedade.

A Figura 6.6 mostra um exemplo desta ocorrência. No exemplo, podemos verificar o alinhamento da propriedade P1 a um traço de execução. Neste traço ocorreram 6 dos 8 eventos declarados na sequência que descreve a propriedade. Dois eventos da sequência foram alinhados a eventos diferentes, ou seja, houve dois mismatches.

Analisando os mismatches podemos verificar, nos destaques circulares, que os eventos 43 e 82 não ocorreram. Porém, o algoritmo, apoiado pelo sistema de pontuação, construído com base na árvore de categorização de eventos, alinhou o evento 43 na propriedade ao evento 44 no traço (coluna 28, linhas 1-3). Da mesma forma, ocorreu o alinhamento entre os eventos 82 e 81 (coluna 24, linhas 7-9). Tanto o par (44,43) como o par (82,81) são extremamente similares no que diz respeito à função. O evento 44 corresponde à requisição de descida a partir do andar 1 enquanto o evento 43 representa a requisição de subida a partir do andar 1. O evento 81 marca a finalização da requisição de descida e o evento 82 marca a finalização da requisição de subida.

Através deste exemplo mostramos que não é necessário definir outra propriedade como, para este exemplo, `44,46,50,52,48,47,82,54` para que o algoritmo de alinhamento local seja capaz de fazer a busca por este cenário semanticamente similar ao cenário original `43,46,50,52,48,47,81,54`. O mesmo não ocorre na busca por strings ou de expressões

regulares em que cada evento (ou conjunto de eventos) a ser buscado deve ser explicitamente declarado.

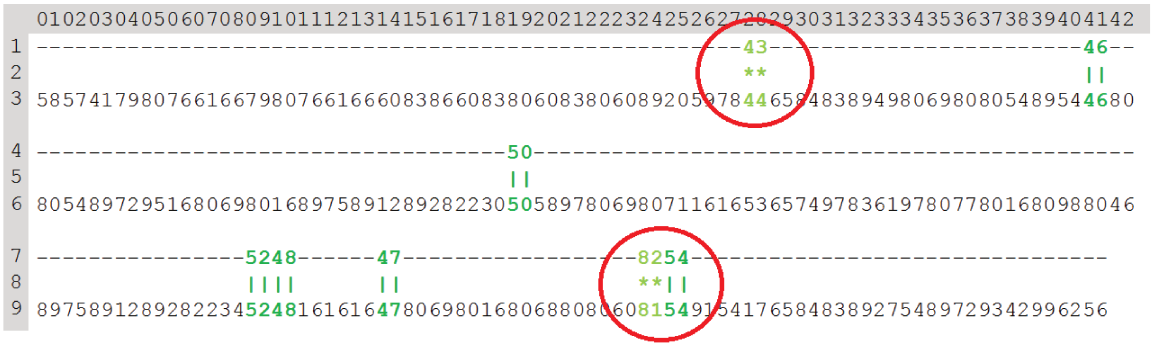


Figura 6.6: Exemplo de ocorrência de eventos semanticamente similares aos definidos na propriedade buscada

6.5 Trabalhos Relacionados

Os autores propuseram, em [19], um abordagem para verificar se traços coletados durante testes de robustez apresentavam comportamentos descritos em invariantes previamente definidas e validadas de acordo com o modelo do SUT. As invariantes são propriedades que expressam robustez e que devem permanecer válidas ao longo de toda a execução do SUT. Os traços são pré-processados e as invariantes são comparadas a eles. Para cada traço analisado, a abordagem obtém um veredicto que informa se o traço falhou ou passou em relação à invariante. A PSSpec foi inspirada na ideia proposta neste trabalho. A principal diferença é o algoritmo utilizado nas abordagens; enquanto a PSSpec utiliza o alinhamento local (matching inexato), Cavalli et al utilizaram um algoritmo de busca exata baseado nos algoritmos KMP [53] e Boyer-Moore [16]. Outra diferença é relacionada ao tipo de propriedade verificada na PSSpec e no trabalho de Cavalli et al. Enquanto este último verifica invariantes que traduzem cenários de robustez, na PSSpec verificamos cenários de não-robustez.

Havelund e Rosu [42, 43] apresentam duas abordagens para verificação, em tempo de execução, de propriedades temporais (descritas com o uso de "past linear temporal logic" [57]) em traços de execução. A primeira abordagem proposta analisa o traço em relação a uma propriedade utilizando o conceito de programação dinâmica que quebra o problema em partes menores para torná-lo mais fácil. Assim, os autores testam se o traço satisfaz cada trecho mais simples da propriedade de forma a verificar se o traço a satisfaz completamente. Na segunda abordagem os autores verificam cada propriedade através do uso de um algoritmo de *rewriting* [69]. Assim, como na PSSpec, os autores verificam propriedades de segurança. Porém, trabalham com propriedades específicas para sistemas de tempo real, diferentemente da PSSpec.

A busca de propriedades também é aplicada na detecção de anomalias. Hangal e Lam [40] desenvolveram a ferramenta Diduce que procura invariantes em implementações de sistemas em tempo de execução. As invariantes buscadas pela ferramenta são propriedades que devem se manter em um ou mais pontos determinados da implementação do sistema analisado e representam aspectos relacionados ao fluxo de dados do sistema como, por exemplo, $x = 0$ sendo x um parâmetro que deve assumir o valor 0 num determinado ponto da execução. A ferramenta Diduce procura apenas por invariantes do tipo <parâmetro - operador - valor>.

A ferramenta Daikon [32, 33], também criada para a detecção de anomalias, detecta a satisfação ou não de invariantes durante a execução de um sistema, examinando os valores que o sistema calcula para determinadas variáveis e determinando padrões de relacionamento entre os valores. As invariantes são definidas a partir de templates de invariantes. Os templates são preenchidos com cada possível variável e o escopo em que ela ocorre na execução do sistema. A partir daí cria-se o conjunto de invariantes a ser procurado.

Chang e Ren, em [20], definem a TBL (*Test Behavior Language*) para especificar propriedades e também para converter traços de execução a fim de compará-los com as propriedades especificadas. A TBL utiliza um formalismo derivado de expressões regulares que agrega o uso de parâmetros e de operadores temporais. Após os traços serem convertidos para a TBL eles são verificados em relação à cada uma das propriedades. A verificação é feita através de um algoritmo baseado em busca de expressões regulares. Como a verificação ocorre em tempo de execução, o volume de informação pode prejudicar o desempenho do algoritmo.

O trabalho de Lo et al [65] consiste do uso de propriedades através de uma abordagem baseada em mineração de processos (*process mining*) para a verificação de traços de execução. O objetivo dos autores é detectar anomalias nos traços. As propriedades, chamadas no trabalho de padrões iterativos (*iterative patterns*) são comportamentos conhecidos, ou seja, sequências de eventos do sistema analisado, utilizados para a construção de um classificador. Este classificador é responsável por buscar, nos traços de execução do sistema, erros não conhecidos. O problema mais comum citado pelos autores é a presença de ruídos, eventos comuns porém não importantes funcionalmente mas que prejudicam o desempenho do classificador.

Em [74], o autor desenvolveu uma abordagem para verificação de invariantes em traços de execução de sistemas web. As invariantes, propriedades que o sistema deve satisfazer, são definidas pelo autor levam em conta a sequência entre as mensagens e o as potenciais restrições de tempo.

Bessayah [12] definiu um oráculo para a verificação de requisitos que representam restrições de tempo real. Tais requisitos são descritos utilizando-se lógica temporal.

Analisando os trabalhos com algum grau de relacionamento com a PSSpec, verificamos alguns pontos em comum em no que diz respeito à PSSpec: a PSSpec é a única destas soluções que mostra visualmente a localização da propriedade no traço analisado e, ainda, é capaz de localizar a ocorrência de cenários funcionalmente similares porém não idênticos à propriedade procurada.

6.6 Considerações Finais

A abordagem PSSpec foi elaborada para complementar a GCSpec. Com a PSSpec, o testador tem a opção de utilizar um oráculo independente da implementação do SUT. Além disso, pode verificar não somente o fluxo de controle como também os parâmetros e seus valores. O algoritmo de alinhamento de sequências oferece como vantagem, em relação à abordagens existentes, a capacidade de localizar não apenas a ocorrência dos eventos procurados mas também a ocorrência de eventos semanticamente similares aos eventos originais (conforme exemplo apresentado na Figura 6.6).

A Tabela 6.4 contém o resumo das diferenças e semelhanças entre as abordagens propostas.

Caracterítica	GCSpec	PSSpec
Oráculo para teste de robustez por meio de injeção de falhas	Sim	Sim
Padrão de comportamento depende da implementação	Sim	Não
Tipo de algoritmo utilizado	Global ou semi-global	Local
Complexidade de tempo do algoritmo - etapa de composição da matriz de similaridades	O(m*n)	O(m*n)
Complexidade de tempo do algoritmo - etapa de recuperação do alinhamento ótimo	O(m+n)	O(m+n)
Utiliza matriz de pontuação baseada na árvore de categorização de eventos	Sim	Sim
Classifica traços avaliados como robustos ou não robustos segundo limiar	Sim	Não
Classifica os traços avaliados desde o considerado menos similar até o considerado como mais similar	Sim	Sim

Tabela 6.4: Comparação entre as características presentes na GCSpec e na PSSpec

onde m e n são os tamanhos das sequências alinhadas em número de símbolos.

Os experimentos quantitativos mostraram que a adoção da penalização atenuada de espaços é vantajosa em relação à penalização constante pois possibilita o alcance, durante o alinhamento, de eventos relevantes para a propriedade que se encontram dispersos no traço, intercalados por eventos comuns porém não interessantes para a sequência buscada.

Com a atenuação das penalidades de espaço não ocorre a perda no valor da pontuação do alinhamento pelo fato que as posições subsequentes num conjunto contíguo de espaços não custam caro ao alinhamento. Na comparação de traços e propriedades, esta é uma ocorrência comum devido à diferença entre os tamanhos de traços e propriedades.

Este fato nos levou também à percepção de que a similaridade calculada de acordo com a métrica proposta na Equação 6.2 e analisada de forma isolada não é adequada como medida

da qualidade do alinhamento obtido pela PSSpec. Apenas quando medimos a quantidade de matches no alinhamento, pudemos verificar de forma inequívoca que a penalização atenuada obtém melhores resultados do que a penalização constante.

Com os experimentos e a análise quantitativa e qualitativa dos resultados obtidos chegamos a algumas diferenças podem ser destacadas quando comparamos a PSSpec à outras abordagens utilizadas em busca de propriedades:

1. Na busca exata, através de algoritmos como KMP [53] e Boyer-Moore [16], a busca se dá por substrings, ou seja, uma sequência de símbolos consecutivos, enquanto que algoritmos de busca inexata, como o alinhamento local se baseiam em busca de subsequências, que podem ter símbolos consecutivos ou não.

2. Na busca por meio de algoritmos de comparação com expressões regulares, o padrão buscado pode variar conforme a expressão descrita. Porém, não há match de símbolos que não tenham sido explicitamente expressos como sendo desejados mesmo que este sejam semanticamente parecidos. A busca ocorre estritamente de acordo com o que é descrito pela expressão regular. A PSSpec é capaz de encontrar símbolos semanticamente similares aos definidos na propriedade quando não há a ocorrência dos eventos procurados no traço avaliado.

3. Na busca por meio de algoritmos de mineração de dados [65] a busca muitas vezes é prejudicada por ruídos, ou seja, pela presença no traço de eventos frequentes porém não importantes do ponto de vista da propriedade. Isto não ocorre na PSSpec pois o algoritmo é guiado pelo sistema de pontuação que agrupa os eventos de acordo com sua similaridade semântica.

4. A PSSPec não precisa de fase de treinmanento ou calibração do algoritmo como ocorre com os algoritmos de mineração (de dados ou de processos).

5. Assim como na GCSpec, na PSSpec a visualização do alinhamento, produto da fase se recuperação do alinhamento, é inerente ao processo e auxília tanto o testador quanto o desenvolvedor na localização do erro e na identificação do contexto no qual ele ocorre;

6. A PSSpec permite a classificação dos traços de execução de acordo com o grau de similaridade entre cada traço e a propriedade analisada. Assim, quanto maior a similaridade maior a chance do traço analisado apresentar não-robustez.

Capítulo 7

Aligner - Implementação das Abordagens GCSpec e PSSpec

A aplicação manual das abordagens propostas envolve uma grande quantidade de esforço para construção da matriz calculada durante o alinhamento dos pares de sequências (M_s). Seria inviável realizarmos as provas de conceito das abordagens GCSpec e PSSpec sem a implementação dos algoritmos local, semi-global e global. Por este motivo, como parte desta pesquisa implementamos uma ferramenta para automatizar os alinhamentos das sequências de teste de robustez.

Na primeira seção descrevemos os requisitos funionais implementados para permitir a aplicação das abordagens propostas. A Seção 7.2 contém a descrição da modelagem da ferramenta e; em seguida, a Seção 7.3 detalha a implementação e a interface da ferramenta. A Seção 7.4 finaliza este capítulo.

7.1 Requisitos Funcionais

Esta seção descreve os requisitos funcionais implementados na ferramenta Aligner:

1. Capacidade de realizar o alinhamento entre duas sequências numéricas que representam:
    - O modelo de comportamento que pode ser descrito pelo padrão-ouro (GCSpec) ou a propriedade descrevendo o cenário de não-robustez (PSSpec), filtrado e codificado;
    - O traço de execução coletado durante o teste de robustez, filtrado e codificado.
2. O alinhamento entre as duas sequências pode ser realizado utilizando-se um dos algoritmos a seguir:
    - Alinhamento local, conforme o algoritmo proposto por Smith e Waterman [92];
    - Alinhamento local com busca por resultados ótimos e próximos a ótimo, conforme algoritmo proposto por Waterman e Eggert [102];

- Alinhamento local com penalidade de espaços atenuada [2],
- Alinhamento semi-global, que não penaliza espaços nas extremidades da sequências envolvidas [91];
- Alinhamento global, conforme algoritmo proposto por Needleman e Wunsch [78].

3. O alinhamento pode ser realizado utilizando-se os seguintes sistemas de pontuação:
   - Constante, no qual o usuário fornece três valores a serem utilizados como pontuação a serem atribuídas a matches, mismatches e espaços;
   - Variável, no qual o usuário fornece para cada par de eventos no alfabeto Σ a pontuação que será utilizada durante o alinhamento.
4. As sequências a serem alinhadas podem ser previamente filtradas a partir do alfabeto de eventos (Σ) a ser fornecido pelo usuário. Como definido no Capítulo 5, $\Sigma = \Sigma_{ev} \cup \Sigma_{er}$ onde Σ_{ev} se refere aos eventos do SUT e Σ_{er} se refere aos eventos de erro presentes no traço.
5. As sequências filtradas podem ser codificadas de acordo com uma função de codificação fornecida pelo usuário. A função de codificação deve apresentar um código numérico e único para cada evento de Σ (conforme definido na Equação 5.1).

7.2 Modelagem da Ferramenta

Na Figura 7.1 são ilustrados os casos de uso da ferramenta do ponto de vista do usuário. A descrição de cada um destes casos de uso encontra-se nas tabelas 7.1, 7.2 e 7.3.

Tabela 7.1: Descrição do caso de uso "Filtrar e codificar sequências de eventos"

Caso de Uso: Filtrar e codificar sequências de eventos	
Pré-condição: -	
Fluxo de Eventos (caminho básico)	Fluxo Secundário (caminho alternativo)
1. O usuário fornece a sequência a ser filtrada e codificada	
2. A sequência de eventos fornecida é válida	2.1. A A sequência de eventos fornecida é inválida
3. O usuário seleciona o alfabeto de eventos do SUT e os códigos a serem atribuídos a cada evento	3.1. O alfabeto e/ou os códigos não são válidos
4. O usuário dispara a filtragem e a codificação da sequência de eventos	

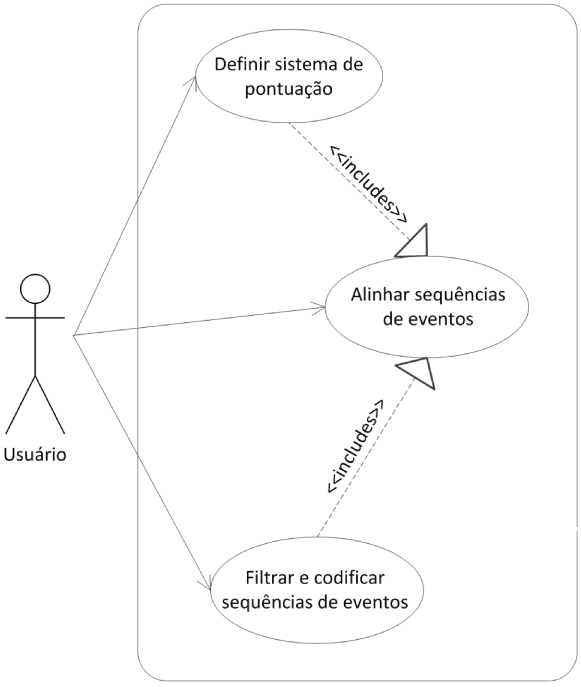


Figura 7.1: Casos de uso da ferramenta Aligner

Tabela 7.2: Descrição do caso de uso "Definir sistema de pontuação"

Caso de Uso: Definir sistema de pontuação	
Pré-condição: O usuário deve estabelecer a matriz de pontuação	
Fluxo de Eventos (caminho básico)	Fluxo Secundário (caminho alternativo)
1. O usuário seleciona o tipo do sistema de pontuação desejado (constante ou variável)	
2. O usuário fornece uma matriz de pontuação válida	2.1. O usuário não fornece a matriz de pontuação ou a matriz é inválida
3. O usuário seleciona em qual SUT o sistema de pontuação deverá ser importado	
4. O usuário importa o sistema de pontuação para a ferramenta Aligner	

A seguir, na Figura 7.2, mostramos o diagrama de classes da ferramenta Aligner. Temos ao todo sete classes, sendo que cinco delas são especializações da classe responsável pelo alinhamento das duas sequências de eventos. Cada uma das especializações implementa o alinhamento de acordo com um dos algoritmos descritos na seção anterior. A última classe contém a modelagem do sistema de pontuação.

Tabela 7.3: Descrição do caso de uso "Alinhar duas sequências de eventos"

Caso de Uso: Alinhar duas sequências de eventos	
Pré-condição 1: O usuário definiu previamente o sistema de pontuação Pré-condição 2: As sequências foram previamente filtradas e codificadas	
Fluxo de Eventos (caminho básico)	Fluxo Secundário (caminho alternativo)
1. O usuário fornece as duas sequências de eventos	1.1. Pelo menos uma das sequências fornecidas é inválida
2. O usuário seleciona o tipo desejado do sistema de pontuação	
3. O usuário seleciona o sistema de pontuação dentre os previamente definidos	
4. O usuário dispara a operação de alinhamento das sequências	

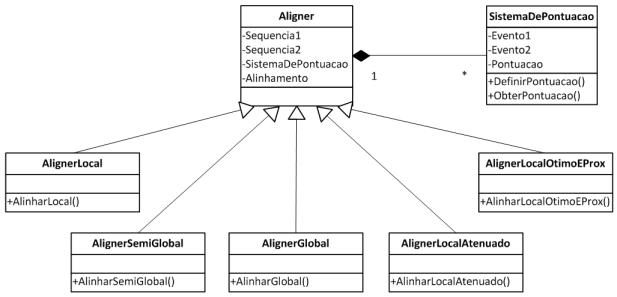


Figura 7.2: Diagrama de classes da ferramenta Aligner

7.3 Detalhamento da Implementação da Ferramenta

Nossa ferramenta foi desenvolvida em como uma aplicação Java[1] (JKD 1.6.0), utilizando Swing através do NetBeans Integrated Development Environment (IDE)[2], versão 7.0.1. A interface da ferramenta Aligner está ilustrada na Figura 7.3. Como a filtragem e a codificação são baseadas no alfabeto de eventos do SUT (Σ) e este alfabeto é específico para cada SUT testado, a funcionalidade foi implementada para os sistemas utilizados durante esta tese. Porém, é possível estendê-la para qualquer sistema alvo de testes de robustez.

O mesmo acontece com o sistema de pontuação variável. Definimos, num banco de dados MySQL[3], as matrizes de pontuação para os sistemas que utilizamos nos estudos de caso. Porém, novas matrizes podem ser inseridas na tabela de matrizes de pontuação. O sistema de pontuação fixo foi estaticamente definido com os valores: match = 4, mismatch = 1 e espaços = -1.

[1] Java é marca registrada da Sun/Oracle

[2] www.netbeans.org

[3] MySQL é marca registrada da Sun/Oracle

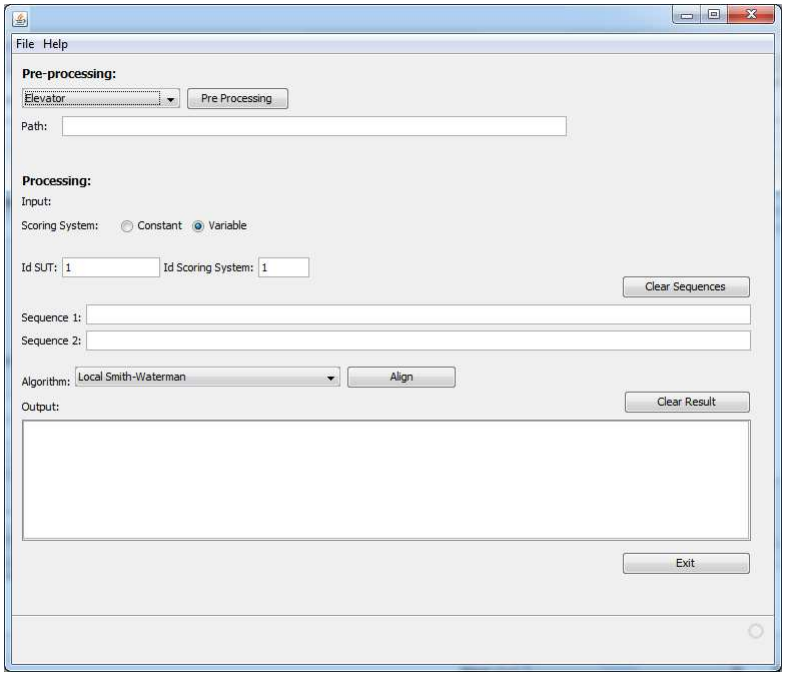


Figura 7.3: Interface da ferramenta Aligner

A Figura 7.4 mostra a saída da execução da ferramenta Aligner.

```
-> File: Execucao_mutElevGroup_AOIS_10_15.txt
Global Alignment - Normal Gaps
-> Score: 1757
-> Position in the similarity matrix: line 389, column 193
Statistics:
-> Matches: 193
-> Mismatches: 00
-> Gaps in the golden run: 196
-> Gaps in the trace: 00
Result:
585741----------7980766166798076616679807661667980766166798076616679807661666083866------60...
||||||          ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||      ||
5857417980766166798076616679807661667980766166798076616679807661666083866083806060...
```

Figura 7.4: Resultado da execução da ferramenta Aligner - Alinhamento de duas sequências

7.4 Considerações Finais

A implementação da ferramenta Aligner teve importância crucial no desenvolvimento desta tese porque as implementações disponíveis na Internet[4][5][6] trabalham com sistemas de pontuação

[4]http://emboss.sourceforge.net/
[5]http://www.ebi.ac.uk/Tools/
[6]http://www.science.co.il/biomedical/sequence-tools.asp

específicos para o alinhamento de sequências biológicas e com alfabetos também restritos a estas sequências.

Assim, mantivemos os algoritmos clássicos de alinhamento de sequências em conjunto com matrizes e codificações de sequências específicas para utilização na avaliação de resultados de teste de robustez.

Porém, faltam ainda melhorias e adequações a serem incorporadas na ferramenta como, por exemplo, a definição de uma interface para entrada de dados das matrizes de pontuação no banco de dados. Atualmente isto tem é feito com o uso direto das ferramentas do MySQL.

Capítulo 8

Conclusões e Trabalhos Futuros

Neste capítulo listamos os problemas relacionados à avaliação de resultados de testes de robustez levantados nesta tese e tecemos as conclusões gerais obtidas a partir do desenvolvimento do trabalho. Além disso, discorremos sobre as limitações existentes nas abordagens e os trabalhos futuros. A Seção 8.1 contém a descrição dos problemas que nos motivaram ao desenvolvimento das abordagens GCSpec e PSSpec e as contribuições de cada uma das abordagens. Na Seção 8.2 elencamos as limitações das abordagens desenvolvidas. O capítulo é finalizado com a Seção 8.3 que apresenta alguns dos possíveis trabalhos futuros que podem ser desenvolvidos a partir desta tese.

8.1 Conclusões Gerais Sobre o Trabalho Desenvolvido

Atualmente, a automatização da coleta e da avaliação dos resultados de testes é tarefa tão importante quanto as demais tarefas que fazem parte do ciclo de vida de teste. Assim, desenvolvemos este trabalho (formado pelas abordagens GCSpec e PSSpec) com o intuito de preencher a lacuna que existe em relação à avaliação de resultados de teste de robustez.

Cada uma das seguintes subseções contém a descrição dos problemas tratados pelas abordagens GCSpec e PSSpec além das contribuições e vantagens de cada uma das abordagens em relação às soluções tradicionais.

8.1.1 GCSpec

A comparação tradicional com o padrão-ouro é uma abordagem bastante utilizada como oráculo não só na avaliação de resultados de teste de conformidade mas também na avaliação de resultados de teste de robustez realizado por meio da técnica de injeção de falhas. Isto ocorre pois a abordagem pode ser aplicada mesmo quando apenas a implementação do SUT encontra-se disponível. Isto é possível pelo fato de que a comparação com o padrão-ouro não utiliza nada além da própria implementação na validação do SUT. Porém, a simplicidade desta abordagem

acarreta consequências nem sempre bem vindas, como por exemplo, uma alta taxa de falsos positivos nos resultados apresentados.

A primeira abordagem desenvolvida neste trabalho, chamada de GCSpec (Golden-run Comparison based on the Specification), estende a comparação de traços de execução coletados durante o teste de robustez com padrões de comportamento. A GCSpec foi proposta visando a diminuição na taxa de falsos positivos apresentados na avaliação de resultados.

Um dos motivos que leva a tradicional comparação com o padrão-ouro ao erro na avaliação de resultados de teste de robustez é o fato desta abordagem utilizar algoritmos de busca exata na comparação. Assim, quando o SUT apresenta comportamento não determinista ou quando mecanismos de tolerância a falhas são disparados tais variações são erroneamente consideradas como sendo defeitos. Outro motivo que pode induzir a comparação tradicional com o padrão-ouro ao erro é a ausência de informações sobre a especificação na avaliação dos traços coletados durante o teste de robustez. Desta forma, o veredicto é baseado apenas na própria implementação, a qual pode apresentar falhas.

A abordagem GCSpec, desenvolvida nesta tese, emprega o algoritmo de alinhamento global (*matching inexato*) na comparação permitindo algumas diferenças entre o traço e o padrão-ouro. Conforme apresentamos nos estudos de caso do Capítulo 5, a aplicação do algoritmo de matching inexato foi adequada tanto para o traço robusto, porém com alguns trechos da execução não exatamente iguais ao padrão de comportamento como para o traço com sinal de não robustez.

A segunda vantagem da GCSpec sobre a comparação tradicional com o padrão-ouro é a incorporação da especificação do SUT na definição da matriz de pontuação utilizada para guiar o posicionamento dos eventos durante o alinhamento. A cada par de eventos a ser alinhado, a matriz de pontuação é consultada para a escolha de pares de eventos semanticamente similares.

Mais uma vantagem da GCSpec é a possibilidade de visualização, através do resultado obtido pelo alinhamento, das posições similares e diferentes entre as sequências alinhadas. Esta visualização auxilia na avaliação do resultado pelo testador e não acarreta em custo computacional extra para o algoritmo pois é inerente a ele sendo obtida na etapa de recuperação do alinhamento ótimo.

8.1.2 PSSpec

Complementar à comparação com o padrão-ouro, a busca por propriedades é outro oráculo utilizado na avaliação de traços de teste de robustez. Dizemos que as abordagens são complementares pois a comparação com o padrão-ouro mostra o quanto o comportamento desvia da situação normal quando o sistema está em presença de falhas, permitindo avaliar se os mecanismos de tolerância a falhas do SUT funcionam adequadamente.

A segunda abordagem criada neste trabalho, chamada de PSSpec (Property Search based on the Specification) estende a tradicional busca por propriedades comparando propriedades de segurança (*safety*) que são buscadas através de cenários de não-robustez em traços de execução coletados durante a injeção de falhas. O objetivo da PSSpec é verificar se tais propriedades são violadas nos traços coletados. Escolhemos empregar o algoritmo de alinhamento local para

a verificação das propriedades pois este algoritmo é empregado para localizar trechos com alta similaridade entre duas sequências o que é adequado à comparação de propriedades e traços pois, comumente, as propriedades apresentam tamanho muito inferior ao dos traços coletados durante os testes.

A busca por propriedades é mais restrita no que se refere ao alcance do oráculo. Inversamente ao que ocorre na comparação com o padrão-ouro, nesta abordagem são buscados cenários que indicam não-robustez derivados das propriedades a serem verificadas. Já, o padrão-ouro indica um comportamento robusto. Assim, na PSSpec, mesmo que a propriedade não seja violada no traço não podemos inferir nenhum resultado sobre sua robusto. Porém, caso um traço comparado ao padrão-ouro obtenha uma similaridade baixa, ele pode ser considerado forte candidato a ser não-robustez. Na PSSpec, não temos um classificador binário que retorne, para cada traço, um veredicto do tipo "robustos/não robustos", mas temos um resultado contínuo, baseado na similaridade entre os traços e a propriedade, a partir do qual é possível criar uma classificação dos traços desde o mais similar ao cenário de não robustez até o menos similar.

Outra característica da busca por propriedades é a possibilidade de haver interferência na busca da propriedade causada por eventos no traço que não são relacionados à propriedade. Algumas vezes, dependendo da quantidade desses eventos, chamados de "ruído", há dificuldade na identificação da propriedade, mesmo que ela esteja presente no traço. Na PSSpec, implementamos a penalização atenuada de espaços de forma que grandes sequências de espaços contíguos recebem penalização menor e não mascarem a pontuação do alinhamento da propriedade ao traço. Associada ao uso da penalização atenuada de espaços, a matriz de pontuação construída com base na árvore de categorização de eventos possibilita a recuperação e posterior visualização de alinhamentos semanticamente significativos que auxiliam o testador no entendimento da situação de não robustez, caso ela esteja presente no traço analisado.

8.2 Limitações das Abordagens Desenvolvidas

Uma das limitações das abordagem GCSpec é a utilização da implementação do SUT para a geração dos padrões-ouro. Neste caso, se a implementação contiver falhas, tais falhas podem ser transmitidas aos traços coletados sem injeção de falhas causando problemas na classificação dos comportamentos como robustos ou não-robustos. Um padrão-ouro com problema de robustez pode ser errôneamente utilizado na análise de resultados de teste de robustez. Este problema não é específico da GCSpec, ele é comum a todas as abordagens que utilizam o próprio SUT para a geração dos traços que servem de padrão de comportamento.

Outra limitação da GCSpec e, neste caso, também da PSSpec é que a avaliação pode ser aplicada apenas após o término dos testes. Não é possível avaliar os resultados durante os testes (de forma online) devido à necessidade de filtragem e codificação dos eventos e também devido à necessidade da coleta do mesmo momento da execução sem e com injeção de falhas (apenas na GCSpec).

A PSSpec não é capaz de identificar propriedades que levem em conta aspectos de tempo

real. Apenas podemos indicar a ordem em que os eventos da propriedade buscada devem ocorrer durante a execução do SUT.

Os traços de execução gerados pelo SUT precisam conter informações sobre os fluxo de controle e os parâmetros (com respectivos valores) para que possam ser avaliados pelas ferramentas desenvolvidas. Caso o SUT não registre o fluxo de controle e de dados nos traços de execução gerados o SUT precisará ser instrumentado de forma que os traços de execução passem a prover tais informações.

8.3 Trabalhos Futuros

O trabalho desenvolvido nesta tese teve foco na análise de resultados de teste de robustez por meio do emprego de algoritmos de alinhamento de sequências. Como dito no Capítulo 3 nos concentramos no estudo e implementação dos algoritmos básicos de alinhamento global, semi-global e local. Porém há inúmeras propostas de otimização dos algoritmos básicos que podem ser estudados para verificar a aplicabilidade e a melhoria nos resultados[49, 10].

Outras estratégias, como o alinhamento de sequências através de uma abordagem probabilística baseada em modelos de Markov tem sido aplicadas na bioinformática. Além disso, abordagens heurísticas [39], largamente aplicadas na bioinformática podem ser estudadas para que os resultados sejam comparados com os obtidos pelos algoritmos básicos em termos de falsos positivos e negativos.

Outra categoria de algoritmos a ser estudada são os alinhamentos de sequências restritivos (*constrained sequence alignment*). Tais algoritmos têm como objetivo verificar se os alinhamentos gerados violam ou mantêm certas propriedades relacionadas ao comportamento semântico das sequências. Tais propriedades podem ser descritas por meio de expressões regulares. Assim, um alinhamento ótimo entre duas sequências só é considerado "aceito" se, além de ter uma pontuação alta, não violar a expressão regular que define a propriedade desejada [5, 22, 71].

Além de diferentes algoritmos, diferentes sistemas de pontuação também podem ser estudados. Altchul desenvolveu uma função de penalização para espaços chamada de generalização de espaços atenuados (*generalized affine gaps*) que pode ser empregada tanto em alinhamentos locais quanto globais sem aumentar a complexidade dos algoritmos básicos. O autor afirma que as tentativas de alinhar otimamente regiões pouco similares acarretam em impacto nas regiões mais similares. Assim, esta função de pontuação é capaz de penalizar diferentemente regiões fortemente relacionadas em relação às regiões dissimilares das sequências.

Melhorias podem ser estudadas também no método de definição da matriz de pontuação para cada SUT, proposto no Capítulo 4. Notamos que, algumas vezes, alguns eventos (principalmente eventos que representam saídas errôneas do SUT) deveriam ser representados em ramos mais "distantes" dos eventos regulares do SUT para que, quando mapeados na matriz de pontuação, recebessem uma pontuação mínima ou próxima da mínima. Uma das formas de se obter este distanciamento é através da da inserção de pesos nas arestas da árvore de categorização de eventos. Os pesos resultarão em maiores distâncias entre os eventos e, por consequência, menores

valores na matriz de pontuação.

Como dissemos na seção anterior, a utilização da própria implementação para geração dos padrões-ouro é uma limitação da ferramenta que pode levar a erros. Assim, outra possibilidade de trabalho futuro é o estudo sobre a geração de padrões-ouro a partir da execução do modelo que descreve o comportamento do SUT.

A utilização das abordagens desenvolvidas para diferentes finalidades é também um tema para trabalhos futuros. A avaliação de benchmarks através da comparação de traços de execução considerando sequências idênticas de entradas pode ser uma aplicação para a GCSpec. Assim como algoritmos de mineração de processos estão sendo aplicados na detecção de anomalias e de defeitos [13, 65], acreditamos também que nossas abordagens também possam ser avaliadas quanto ao desempenho nestas áreas.

Referências Bibliográficas

[1] T. Al-Ibaisi, A. El-Latif, A. Dalhoum, M. Al-Rawi, M. Alfonseca, and A. Ortega. Network intrusion detection using genetic algorithm to find best dna signature. *WSEAS Transactions on Systems*, 7(7):589–599, 2008.

[2] S.F. Altschul. Generalized affine gap costs for protein sequence alignment. *Proteins*, 32:88–96, 1989.

[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. A basic local alignment search tool. *J. Mol. Biol*, 215:403–410, 1990.

[4] J.H. Andrews. Testing using log file analysis: Tools, methods, and issues. In *Proceedings of the 22nd 13th IEEE International Conference on Advanced Software Engineering*, pages 157–166, Honolulu, Hawaii, 1998.

[5] A.N. Arslan. Regular expression constrained sequence alignment. *Journal of Discrete Algorithms*, 5(4):647–661, 2007.

[6] P. Avgustinov, J. Tibble, and O. Moor. Making trace monitors feasible. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications*, pages 589–608, Montreal, Canada, 2007.

[7] A. Avizienis, J-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[8] D. Avresky, J. Arlat, J.C. Laprie, and Y. Crouzet. Fault injection for the formal testing of fault tolerance. In *Proceedings of the 22nd International Symposium on Fault-Tolerant Computing*, pages 345–354, Boston, MA, USA, 1992.

[9] L. Baresi and M. Young. Test oracles. Technical report, University of Oregon, Department of Computer and Information Science, 2001. CIS-TR-01-02.

[10] G.J. Barton. An efficient algorithm to locate all locally optimal alignments between two sequences allowing for gaps. *Computer Applications in the Biosciences: CABIOS*, 9(6):729–734, 1993.

[11] E. Bayse, A. Cavalli, M. Nunez, and F. Zaidi. A passive testing approach based on invariants: Application to the wap. *Computer Networks*, 48(2):247–266, 2005.

[12] F. Bessayah. *Une Approche Complémentaire de Test de Robustesse Basée sur lInjection de Fautes et le Test Passif*. PhD thesis, Télécom e Management SudParis and Université Pierre et Marie Curie - Paris 6, 2010.

[13] F.L. Bezerra. *Algoritmos de Detecção de Anomalias em Logs de Sistemas Baseados em Processos de Negócios*. PhD thesis, Universidade Estadual de Campinas, 2011.

[14] R.V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley Professional, 1999.

[15] R.P.J.C. Bose and W.M.P. van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Information Systems*, 37(2):117–141, April 2012.

[16] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[17] A.C.S. Braga. *Curvas ROC: Aspectos Funcionais e Aplicações*. PhD thesis, Universidade do Minho, 2000.

[18] A. Cain, T.Y. Chen, D. Grant, P.L. Poon, S.F. Tang, and T.H. Tse. Addict: A prototype system for automated test data generation using the integrated classification-tree methodology. In *In Proceedings of the 1st ACIS International Conference on Software Engineering Research and Applications*, pages 225–238, December 2004.

[19] A. Cavalli, E. Martins, and A.N.P. Morais. Use of invariant properties to evaluate the results of fault-injection-based robustness testing of protocol implementations. In *In Proceedings of the 4th Workshop on Advances in Model Based Testing, joint with 1st IEEE International Conference on Software Testing*, pages 21–30, 2008.

[20] F. Chang and J. Ren. Validating system properties exhibited in execution traces. In *Proceedings of the 22th IEEE/ACM International Conference on Automated software engineering*, pages 517–520, Atlanta, Georgia, USA, 2007.

[21] K. Chao and L. Zhang. *Sequence Comparison: Theory and Methods*. Springer, 2009.

[22] Y. Chung, C.L. Lu, and C. Tang. Constrained sequence alignment: A general model and the hardness results. *Discrete Applied Mathematics*, 155:2471–2486, 2007.

[23] J. Cohen, D. Plakosh, and K. Keeler. Robustness testing of software-intensive systems: Explanation and guide. Technical report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2005. Technical Note - CMU/SEI-2005-TN-015.

[24] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[25] S. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *In Proceedings of the 19th Annual Computer Security Applications Conference*, pages 24–33, Las Vegas, NV, USA, December 2003.

[26] S.E. Coull and B.K. Szymanski. Sequence alignment for masquerade detection. *Computational Statistics and Data Analysis*, 52:4116–4131, 2008.

[27] W.H.E. Day and H. Edelsbrunner. Efficient algorithms for agglomerative hierarchical clustering methods. *Journal of Classification*, 1:7–24, 1984.

[28] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. Amino acid substitution matrices from protein blocks. *Atlas of Protein Sequences and Structure*, 5:345–352, 1978.

[29] M.E. Delamaro, J.C. Maldonado, and M. Jino. *Introdução ao Teste de Software*. Elsevier, 2007.

[30] J.A. Duraes and H.S. Madeira. Emulation of software faults: A field data study and a practical approach. *IEEE Transactions on Software Engineering*, 32(11):849–867, November 2006.

[31] R.C. Edgar. Optimizing substitution matrix choice and gap parameters for sequence alignment. *BMC Bioinformatics*, 10:396–407, 2009.

[32] M.D. Ernst, J. Cockrell, W.G. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. *Electronic Notes in Theoretical Computer Science*, 55(2):255–276, 2001.

[33] M.D. Ernst, J.H. Perkins, P.J. Guo, S. McCamant, C. Pacheco, M.S. Tschantz, and C. Xiao. The daikon system for dynamic detection of likely invariants. *Science of computer programming*, 69:35–45, 2007.

[34] T. Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[35] A. Ghosh and M. Schmid. An approach to testing cots software for robustness to operating system exceptions and errors. In *Proceedings of 10th International Symposium on Software Reliability Engineering*, pages 166–174, Boca Raton, FL, USA, 1999.

[36] O. Gotoh. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, 162:705–708, December 1982.

[37] M. Grochtmann and K. Grimm. Classification trees for partition testing. *Software Testing, Verification and Reliability*, 3(2):63–82, 1993.

[38] R. Gupta, R. Sharma, and S. Rawat. Review paper on software testing. In *Proceedings of the 2012 International Conference on Computer Applications (ICCA'12)*, Pondicherry, India, 2012.

[39] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[40] S. Hangal and M.S. Lam. Tracking down software bugs using automatic anomaly detection. In *Proceedings of the 24th International Conference on Software Engineering*, pages 291–301, Orlando, FL, USA, 2002.

[41] W. Haque, A. Aravind, and B. Reddy. Pairwise sequence alignment algorithms - a survey. In *Proceedings of the 2009 Conference on Information Science, Technology and Applications (ISTA'09)*, Kwait City, Kwait, 2009.

[42] K. Havelund and G. Rosu. Synthesizing monitors for safety properties. *Lecture Notes in Computer Science*, 2280:257–268, 2002.

[43] K. Havelund and G. Rosu. Efficient monitoring of safety properties. *International Journal on Software Tools for Technology Transfer*, 6:158–173, 2004.

[44] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Biochemistry*, 89:10915–10919, November 1992.

[45] M. Hiller, A. Jhumka, and N. Suri. An approach for analysing the propagation of data errors in software. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 161–172, Göteborg, Sweden, 2001.

[46] W.E. Howden. Theoretical and empirical studies of program testing. In *Proceedings of the 3rd International Conference on Software Engineering*, Atlanta, Georgia, USA, 1978.

[47] X. Huang. Sequence alignment with an appropriate substitution matrix. *Journal of Computational Biology*, 15(2):396–407, 2008.

[48] X. Huang and K.M. Chao. A generalized global alignment algorithm. *Bioinformatics*, 19:228–233, 2003.

[49] X. Huang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12(2):337–357, September 1991.

[50] IEEE. Ieee standard glossary of software engineering terminology. Technical report, Institute of Electrical and Electronic Engineers, New York, USA, 1990. IEEE Std 610.12–1990.

[51] M.Y. Kao. *Encyclopedia of Algorithms*. Springer, 2008.

[52] M. Kessentini, H. Sahraoui, and M. Boukadoum. Example-based model-transformation testing. *Automated Software Engneering*, 18(2):199–224, 2011.

[53] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[54] N.P. Kropp, P. Koopman, and D. Siewiorek. Automated robustness testing of off-the-shelf software components. In *Proceedings of the Twenty-Eigth Annual International Symposium on Fault-Tolerant Computing*, pages 230–239, June 1998.

[55] L. Lamport. Providing the correctness of multiprocess programs. *IEEE Transaction on Software Engineering*, SE-3(2):124–143, 1977.

[56] N. Laranjeiro, R. Oliveira, and M. Vieira. Applying text classification algorithms in web services robustness testing. In *Proceedings of the 29th IEEE Simposium on Reliable Distributed Systems*, November 2009.

[57] T. Latvala, A. Biere, K. Heljanko, and T. Juntti. Simple is better: Efficient bounded model checking for past ltl. In *Proceedings of the 6th International Conference Verification, Model Checking, and Abstract Interpretation*, pages 380–395, Paris, France, 2005.

[58] M. Leeke and A. Jhumka. Evaluating the use of reference run models in fault injection analysis. In *Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 16–18, Shangai, China, 2009.

[59] E. Lehmann and J. Wegener. Test case design by means of the cte xl. In *In Proceedings of the 8th European International Conference on Software Testing, Analysis e Review (EuroSTAR 2000)*, Kopenhagen, Denmark, December 2000.

[60] B. Lei, X. Li, Z. Liu, C. Morisset, and V. Stolz. Robustness testing for software components. *Science of Computer Programming*, 75(10):879–897, 2010.

[61] G.S. Lemos and E. Martins. Robustness testing oracle using a sequence alignment algorithm. In *In Proceedings of The First Workshop on Software Test Output Validation (STOV) - Co-located with International Symposium on Software Testing and Analysis*, Trento, Italy, July 2010.

[62] G.S. Lemos and E. Martins. Detecting non-robust behavior: A bioinformatics approach. In *In Proceedings of The 41st International Conference on Dependable Systems and Networks. Fast Abstract*, Hong Kong, China, June 2011.

[63] G.S. Lemos and E. Martins. Specification-guided golden run for analysis of robustness testing results. In *In Proceedings of The Sixth International Conference on Software Security and Reliability*, Gaithersburg, MA, USA, June 2012.

[64] N.G. Leveson. Software safety: Why, what, and how. *ACM Computing Surveys*, 18(2):125–163, June 1986.

[65] D. Lo, J. Han, H. Cheng, S.C. Khoo, and C. Sun. Classification of software behaviors for failure detection: A discriminative pattern mining approach. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining*, pages 557–566, Paris, France, July 2009.

[66] P.D.L. Machado. *Testing from Structured Algebraic Specifications: The Oracle Problem*. PhD thesis, University of Edinburgh, 2000.

[67] Y.A. Markov and M.O. Kalinin. Intellectual intrusion detection with sequences alignment methods. In *Proceedings of the 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security*, pages 217–228, San Petersburg, Russia, September 2010.

[68] E. Martins, C.M.F. Rubira, and N.G.M. Leme. Jaca: a reflective fault injection tool based on patterns. In *Proceedings of The 32th International Conference on Dependable Systems and Networks*, pages 483–487, Bethesda, Maryland, USA, 2002.

[69] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96:73–155, 1992.

[70] W. Miller and E.W. Myers. Sequence comparison with concave weighting functions. *Bulletin of Mathematical Biology*, 50(2):97–120, 1988.

[71] N. Milo, T. Pinhas, and M. Ziv-Ukelson. Sa-repc sequence alignment with regular expression path constraint. *Lecture Notes in Computer Science*, 6031:451–462, 2010.

[72] A.V. Moorsel and H. Madeira. Amber consortium report - state of the art in resilience assessment, measurement and benchmarking. Technical report, University of Coimbra, 2009.

[73] R. Moraes, R. Barbosa, J. Duraes, N. Mendes, E. Martins, and H. Madeira. Injection of faults at component interfaces and inside the component code: Are they equivalent? In *Proceedings of The 33th International Conference on Software Engineering*, pages 53–64, Coimbra, Portugal, October 2006.

[74] G. Morales. *A test methodology for the validation of web applications*. PhD thesis, Télécom e Management SudParis and Université Pierre et Marie Curie - Paris 6, 2010.

[75] D.W. Mount. *Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, 2nd edition edition, 2004.

[76] E.W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Biosciences*, 4(1):11–17, 1988.

[77] D. Naor and D.L. Brulag. On near-optimal alignment of biological sequences. *Journal of Computational Biology*, 1(4):349–366, 1994.

[78] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[79] S.O. Ogata. Alinhamento de sequências biológicas com o uso de algoritmos genéticos. Master's thesis, Centro de Ciências Biológicas e da Saúde, UFSCar, 2005.

[80] K.M. Olender and L.J. Osterweil. Cecil: A sequencing constraint language for automatic static analysis generation. *IEEE Transactions on Software Engineering*, 16(3):268–280, March 1990.

[81] T.J. Ostrand and M.J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, 1988.

[82] W.R. Pearson and D.J. Lipman. Improved tools for biological sequence comparison, 1998.

[83] P. Pelliccione, H. Muccini, N. Guelfi, and A. Romanovsky. *Software Engineering of Fault Tolerant Systems*, volume 19. Series on Software Engineering and Knowledge Engineering, 2007.

[84] R.C. Prati, G.E.A.P.A. Batista, and M.C.Monard. Curvas roc para avaliação de classificadores. *IEEE América Latina*, 6(2), 2008.

[85] J.T. Reese and W.R. Pearson. Empirical determination of effective gap penalties for sequence comparison. *Bioinformatics*, 18(11):1500–1507, 2002.

[86] D.J. Richardson, S.L. Aha, and T.O. O'Malley. Specification-based test orcles for reactive systems. In *Proceedings of the 14th International Conference on Software Engineering*, pages 105–118, Melbourne, Australia, May 1992.

[87] A. Rollet. Testing robustness of real-time embedded systems. In *Proceedings of the Workshop on Testing Real-Time and Embedded Systems*, Pisa, Italy, 2003.

[88] F. Saad-Khorchef, I. Berrada, A. Rollet, and R. Castanet. Automated robustness testing for reactive systems: Application to communicating protocols. In *In Proceedings of the Innovative Internet Computing Systems - IICS'10*, pages 409–421, Bangkok, Thailand, June 2010.

[89] F. Saad-Khorchef, A. Rollet, and R. Castanet. A framework and a tool for robustness testing of communicating software. In *Proceedings of the 22nd Annual ACM Symposium on Applied Computing*, pages 345–354, Seoul, Korea, March 2007.

[90] F. Salfner, M. Lenk, and M. Malek. A survey of online failure prediction methods. *ACM Computing Surveys*, 42(9):1–42, 2010.

[91] J.C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* PWS Publishing Company, 1997.

[92] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[93] A.S. Sodiya, O. Folorunso, S.A. Onashoga, and O.P. Ogunderu. An improved semi-global alignment algorithm for masquerade detection. *International Journal of Network Security*, 12(3):211–220, May 2011.

[94] I. Sommerville. *Software Engineering*. International Computer Science Series. Addison-Wesley, 2006.

[95] K.T. Tseng, C.B. Yang, K.S. Huang, and Y.H. Peng. Near-optimal block alignments. *IEICE Transactions on Information and Systems*, pages 789–795, 2008. Volume E91-D(3).

[96] H.T. Uyar, A.S. Uyar, and E. Harmanci. Pairwise sequence comparison for fitness evaluation in evolutionary structural software testing. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1959–1960, Seattle, Washington, USA, July 2006.

[97] M. Vieira, N. Laranjeiro, and H. Madeira. Benchmarking the robustness of web services. In *Proceedings of The 13th International Symposium on Pacific Rim Dependable Computing*, pages 322–329, December 2007.

[98] M. Vingron. Near-optimal sequence alignment. *Current Opinion in Structural Biology*, 6(3):346–352, June 1996.

[99] WAPForum. Wireless transaction protocol specification. Technical report, WAP Forum, 1999. SPEC-WTP-19990611.pdf.

[100] WAPForum. Wireless application protocol architecture. Technical report, WAP Forum, 2001. wap-210-waparch-20010712-a.pdf.

[101] M. Waterman. Efficient sequence alignment algorithms. *Journal of Theoretical Biology*, 108:333–337, 1984.

[102] M.S. Waterman and M. Eggert. A new algorithm for best subsequence alignments with application to trna-rrna comparisons. *Journal of Molecular Biology*, 197:723–728, 1987.

[103] S. Winter, C. Sarbu, N. Suri, and B. Murphy. The impact of fault models on software robustness evaluations. In *Proceedings of The 33th International Conference on Software Engineering*, Honolulu, Hawaii, USA, May 2011.

[104] C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: an Introduction*. Kluwer Academic Publishers, 2000.

[105] Q. Xie and A. M. Memon. Designing and comparing automated test oracles for gui-based software applications. *ACM Transactions on Software Engineering and Methodology*, 16(1), February 2007.

[106] M. Zucker. Suboptimal sequence alignment in molecular biology. *Journal of Molecular Biology*, 221:403–420, 1989.

Apêndice A

Listagem dos Algoritmos Básicos de Programação Dinâmica

A seguir, nas Seções A.1, A.2 e A.3 estão os três principais algoritmos de alinhamento de sequências baseados em programação dinâmica. Todos são O compostos de duas etapas: a definição da matriz de similaridades com os alinhamentos entre as sequências e a recuperação do alinhamento ótimo a partir da matriz construída na etapa anterior. A Seção A.4 contém outros algoritmos complementares.

A.1 Algoritmo de Alinhamento Global

A Listagem A.1 descreve a implementação, em Java, da primeira etapa do algoritmo de alinhamento de pares de sequências global. Esta etapa é responsável pela definição da matriz de similaridades. Já, a Listagem A.2 contém a segunda etapa do algoritmo, que recupera o alinhamento global ótimo entre as duas sequências a partir da matriz construída na etapa anterior.

Listing A.1: Definição da matriz de similaridades - algoritmo de alinhamento global

```
int[][] calculate_Global_SimilarityMt(int[][] ScoringMt,
                                      int[] Sequence1, int[] Sequence2)
{
  // Define new similarity matrix
  int[][] SimilarityMt = new int[Sequence2.length+1][Sequence1.length+1];

  // Fill first line and first column with zeros
  for (int line = 0; line < Sequence2.length + 1; line++)
  {
     SimilarityMt[line][0] = line*ScoringMt[1][Sequence2[line-1]];
  }
  for (int column = 0; column < Sequence1.length + 1; column++)
  {
     SimilarityMt[0][column] = column*ScoringMt[Sequence1[column-1]][1];
  }
```

```
  // Fill others positions
  for (int line = 1; line < Sequence2.length + 1; line++)
  {
     for (int column = 1; column < Sequence1.length + 1; column++)
     {
        // Obtain scores from scoring_matrix
        int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
        int vertical_score = ScoringMt[1][Sequence2[line-1]];
        int horizontal_score = ScoringMt[Sequence1[column-1]][1];

        // Add with previous calculations
        int total_diagonal = SimilarityMt[line-1][column-1] + diagonal_score;
        int total_horizontal = SimilarityMt[line][column-1] + horizontal_score;
        int total_vertical  = SimilarityMt[line-1][column] + vertical_score;

        // Define maximum score
        int max_score = Math.max(total_diagonal, total_horizontal);
        SimilarityMt[line][column] = Math.max(max_score,total_vertical);
     }
  }

  return SimilarityMt;
}
```

Listing A.2: Recuperação do alinhamento ótimo - algoritmo de alinhamento global

```
String print_GlobalAlignment(int[][] SimilarityMt, int[][] ScoringMt,
                             int[] Sequence1, int[] Sequence2)
{
  // Initialize variables
  String string_Best_Alignment = "";
  String string_Result_Line1 = "";
  String string_Result_Line2 = "";

  // Obtain max position
  int column = Sequence1.length;
  int line = Sequence2.length;

  // Obtain position of backtracking start in the similarity matrix
  Matrix_cell max_Matrix_cell = new Matrix_cell();
  max_Matrix_cell.setValue(SimilarityMt[column][line]);
  max_Matrix_cell.setLine(line);
  max_Matrix_cell.setColumn(column);
  int startLine = line;
  int startColumn = column;

  // Fill with gaps before alignment
  while (line > startLine)
  {
     strResline1 = "-" + strResline1;
     string_Result_Line2 = Integer.toString(Sequence2[line-1])
                           + string_Result_Line2;
     line--;
  }
  while (column > startColumn)
  {
     strResline1 = Integer.toString(Sequence1[column-1]) + strResline1;
     string_Result_Line2 = "-" + string_Result_Line2;
```

```
    column--;
}

// Other backtracking positions calculation
while ((line > 0) && (column > 0)) {

    // Initialize variables
    int current_score = SimilarityMt[line][column];

    int current_diagonal_score = SimilarityMt[line-1][column-1];
    int current_vertical_score = SimilarityMt[line-1][column];
    int current_horizontal_score = SimilarityMt[line][column-1];

    int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
    int vertical_score = ScoringMt[1][Sequence2[line-1]];
    int horizontal_score = ScoringMt[Sequence1[column-1][1]];

    // Find out from where the score was obtained
    if (current_score == current_diagonal_score + diagonal_score)
    {
        string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
        string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
        line--;
        column--;
    }
    else if (current_score == current_vertical_score + vertical_score)
         {
            string_Result_Line1 = "-" + string_Result_Line1;
            string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
            line--;
         }
         else if (current_score == current_horizontal_score + horizontal_score)
              {
                 string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
                 string_Result_Line2 = "-" + string_Result_Line2;
                 column--;
              }
}

// Fill last positions with zeros
while (line > 0)
{
    string_Result_Line1 = "-" + string_Result_Line1;
    string_Result_Line2 = Integer.toString(Sequence2[line-1])
                          + string_Result_Line2;
    line--;
}
while (column > 0)
{
  string_Result_Line1 = Integer.toString(Sequence1[column-1])
                        + string_Result_Line1;
  string_Result_Line2 = "-" + string_Result_Line2;
  column--;
}

string_Best_Alignment = string_Result_Line1 + "\n"
                        + string_Result_Line2 + "\n";
return string_Best_Alignment;
```

```
}
```

A.2 Algoritmo de Alinhamento Semi-Global

A Listagem A.3 descreve a implementação, em Java, da primeira etapa do algoritmo de alinhamento de pares de sequências semi-global. Esta etapa é responsável pela definição da matriz de similaridades. Já, a Listagem A.4 contém a segunda etapa do algoritmo, que recupera o alinhamento semi-global ótimo entre as duas sequências a partir da matriz construída na etapa anterior.

Listing A.3: Definição da matriz de similaridades - algoritmo de alinhamento semi-global

```
int[][] calculate_SemiGlobal_SimilarityMt(int[][] ScoringMt,
                                          int[] Sequence1, int[] Sequence2)
{
  // Define new similarity matrix
  int[][] SimilarityMt = new int[Sequence2.length+1][Sequence1.length+1];

  // Fill first line and first column with zeros
  for (int line = 0; line < Sequence2.length + 1; line++)
  {
     SimilarityMt[line][0] = 0;
  }
  for (int column = 0; column < Sequence1.length + 1; column++)
  {
     SimilarityMt[0][column] = 0;
  }

  // Fill others positions
  for (int line = 1; line < Sequence2.length + 1; line++)
  {
     for (int column = 1; column < Sequence1.length + 1; column++)
     {
        // Obtain scores from scoring_matrix
        int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
        int vertical_score = ScoringMt[1][Sequence2[line-1]];
        int horizontal_score = ScoringMt[Sequence1[column-1]][1];

        // Add with previous calculations
        int total_diagonal = SimilarityMt[line-1][column-1] + diagonal_score;
        int total_horizontal = SimilarityMt[line][column-1] + horizontal_score;
        int total_vertical  = SimilarityMt[line-1][column] + vertical_score;

        // Define maximum score
        int max_score = Math.max(total_diagonal, total_horizontal);
```

```
            SimilarityMt[line][column] = Math.max(max_score, total_vertical);
        }
    }

    return SimilarityMt;
}
```

Listing A.4: Recuperação do alinhamento semi-global ótimo

```
String print_SemiGlobal_Alignment(int[][] SimilarityMt, int[][] ScoringMt,
                                  int[] Sequence1, int[] Sequence2)
{
    // Initialize variables
    String string_Best_Alignment = "";
    String string_Result_Line1 = "";
    String string_Result_Line2 = "";

    // Obtain max position
    int column = Sequence1.length;
    int line = Sequence2.length;

    // Obtain position of backtracking start in the similarity matrix
    Matrix_cell max_Matrix_cell = new Matrix_cell();
    max_Matrix_cell = locateSemiGlobalMaxMatrix(column, line, similarityMt);
    int startLine = max_Matrix_cell.getLine();
    int startColumn = max_Matrix_cell.getColumn();

    // Fill with gaps before alignment
    while (line > startLine)
    {
        strResline1 = "-" + strResline1;
        string_Result_Line2 = Integer.toString(Sequence2[line - 1])
                              + string_Result_Line2;
        line--;
    }
    while (column > startColumn)
    {
        strResline1 = Integer.toString(Sequence1[column-1]) + strResline1;
        string_Result_Line2 = "-" + string_Result_Line2;
        column--;
    }

    // Other backtracking positions calculation
    while ((line > 0) && (column > 0)) {
```

```
    // Initialize variables
    int current_score = SimilarityMt[line][column];

    int current_diagonal_score = SimilarityMt[line-1][column-1];
    int current_vertical_score = SimilarityMt[line-1][column];
    int current_horizontal_score = SimilarityMt[line][column-1];

    int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
    int vertical_score = ScoringMt[1][Sequence2[line-1]];
    int horizontal_score = ScoringMt[Sequence1[column-1][1]];

    // Find out from where the score was obtained
    if (current_score == current_diagonal_score + diagonal_score)
    {
       string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
       string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
       line--;
       column--;
    }
    else if (current_score == current_vertical_score + vertical_score)
         {
            string_Result_Line1 = "-" + string_Result_Line1;
            string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
            line--;
         }
         else if (current_score == current_horizontal_score + horizontal_score)
              {
                 string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
                 string_Result_Line2 = "-" + string_Result_Line2;
                 column--;
              }
}

// Fill last positions with zeros
while (line > 0)
{
   string_Result_Line1 = "-" + string_Result_Line1;
   string_Result_Line2 = Integer.toString(Sequence2[line-1])
                         + string_Result_Line2;
   line--;
}
while (column > 0)
{
  string_Result_Line1 = Integer.toString(Sequence1[column-1])
                        + string_Result_Line1;
  string_Result_Line2 = "-" + string_Result_Line2;
```

```
    column--;
  }

  string_Best_Alignment = string_Result_Line1 + "\n"
                        + string_Result_Line2 + "\n";
  return string_Best_Alignment;
}
```

A.3 Algoritmo de Alinhamento Local

A Listagem A.5 descreve a implementação, em Java, da primeira etapa do algoritmo de alinhamento local de pares de sequências, responsável pela definição da matriz de similaridades. Já, a Listagem A.6 contém a segunda etapa do algoritmo, que recupera o alinhamento ótimo entre as duas sequências a partir da matriz construída na etapa anterior.

Listing A.5: Definição da matriz de similaridades - algoritmo de alinhamento local

```
int[][] calculate_Local_SimilarityMt(int[][] ScoringMt,
                                     int[] Sequence1, int[] Sequence2)
{
  // Define new similarity matrix
  int[][] SimilarityMt = new int[Sequence2.length+1][Sequence1.length+1];

  // Fill first line and first column with zeros
  for (int line = 0; line < Sequence2.length + 1; line++)
  {
     SimilarityMt[line][0] = 0;
  }
  for (int column = 0; column < Sequence1.length + 1; column++)
  {
     SimilarityMt[0][column] = 0;
  }

  // Fill others positions
  for (int line = 1; line < Sequence2.length + 1; line++)
  {
     for (int column = 1; column < Sequence1.length + 1; column++)
     {
        // Obtain scores from scoring_matrix
        int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
        int vertical_score = ScoringMt[1][Sequence2[line-1]];
        int horizontal_score = ScoringMt[Sequence1[column-1]][1];

        // Add with previous calculations
        int total_diagonal = SimilarityMt[line-1][column-1] + diagonal_score;
```

```
        int total_horizontal = SimilarityMt[line][column-1] + horizontal_score;
        int total_vertical   = SimilarityMt[line-1][column] + vertical_score;

        // Define maximum score
        int max_score = Math.max(total_diagonal, total_horizontal);
        max_score = Math.max(max_score,total_vertical);
        SimilarityMt[line][column] = Math.max(max_score,0);
      }
    }

    return SimilarityMt;
}
```

Listing A.6: Recuperação do alinhamento ótimo - algoritmo de alinhamento local

```
String print_LocalAlignment(int[][] SimilarityMt, int[][] ScoringMt,
                            int[] Sequence1, int[] Sequence2)
{
    // Initialize variables
    String string_Best_Alignment = "";
    String string_Result_Line1 = "";
    String string_Result_Line2 = "";

    // Obtain max position
    int column = Sequence1.length;
    int line = Sequence2.length;

    // Obtain position of backtracking start in the similarity matrix
    Matrix_cell max_Matrix_cell = new Matrix_cell();
    max_Matrix_cell = locateLocalMaxMatrix(column, line, similarityMt);
    int startLine = max_Matrix_cell.getLine();
    int startColumn = max_Matrix_cell.getColumn();

    // Fill with gaps before alignment
    while (line > startLine)
    {
        strResline1 = "-" + strResline1;
        string_Result_Line2 = Integer.toString(Sequence2[line-1])
                              + string_Result_Line2;
        line--;
    }
    while (column > startColumn)
    {
        strResline1 = Integer.toString(Sequence1[column-1]) + strResline1;
        string_Result_Line2 = "-" + string_Result_Line2;
        column--;
```

```
}

// Other backtracking positions calculation
while ((line > 0) && (column > 0)) {

   // Initialize variables
   int current_score = SimilarityMt[line][column];

   int current_diagonal_score = SimilarityMt[line-1][column-1];
   int current_vertical_score = SimilarityMt[line-1][column];
   int current_horizontal_score = SimilarityMt[line][column-1];

   int diagonal_score = ScoringMt[Sequence1[column-1]][Sequence2[line-1]];
   int vertical_score = ScoringMt[1][Sequence2[line-1]];
   int horizontal_score = ScoringMt[Sequence1[column-1][1]];

   // Find out from where the score was obtained
   if (current_score == current_diagonal_score + diagonal_score)
   {
      string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
      string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
      line--;
      column--;
   }
   else if (current_score == current_vertical_score + vertical_score)
        {
           string_Result_Line1 = "-" + string_Result_Line1;
           string_Result_Line2 = Sequence2[line-1] + string_Result_Line2;
           line--;
        }
        else if (current_score == current_horizontal_score + horizontal_score)
             {
                string_Result_Line1 = Sequence1[column-1] + string_Result_Line1;
                string_Result_Line2 = "-" + string_Result_Line2;
                column--;
             }
}

// Fill last positions with zeros
while (line > 0)
{
   string_Result_Line1 = "-" + string_Result_Line1;
   string_Result_Line2 = Integer.toString(Sequence2[line-1])
                         + string_Result_Line2;
   line--;
}
```

```
  while (column > 0)
  {
    string_Result_Line1 = Integer.toString(Sequence1[column-1])
                          + string_Result_Line1;
    string_Result_Line2 = "-" + string_Result_Line2;
    column--;
  }

  string_Best_Alignment = string_Result_Line1 + "\n"
                          + string_Result_Line2 + "\n";
  return string_Best_Alignment;
}
```

A.4 Algoritmos Complementares

Esta seção contém as funções complementares utilizadas pelos algoritmos de alinhamento de sequências (A.7).

Listing A.7: Funções complementares

```
Matrix_cell locateLocalMaxMatrix(int lenghtSeq1, int lenghtSeq2,
                                 int[][] SimilarityMt)
{
   // Initialize vars
   int line = 0;
   int temp_line = 0;
   int column = 0;
   int temp_column = 0;
   int maxValue_cellMatrix = -1;

   // Looking for the maximum cell in the similarity matrix
   for (line = 0; line <= lenghtSeq2; line++)
   {
      for (column = 0; column <= lenghtSeq1; column++)
      {
         if (maxValue_cellMatrix < SimilarityMt[line][column])
         {
             maxValue_cellMatrix = SimilarityMt[line][column];
             temp_line = line;
             temp_column = column;
         }
      }
   }

   // Set output value
```

```
    Matrix_cell maxValue_cell = new Matrix_cell();
    maxValue_cell.setValMatrixElement(maxValue_cellMatrix);
    maxValue_cell.setPosi(temp_line);
    maxValue_cell.setPosj(temp_column);

    return maxValue_cell;
}

Matrix_cell locateSemiGlobalMaxMatrix(int lenghtSeq1, int lenghtSeq2,
                                      int[][][] SimilarityMt)
{
    // Initialize vars
    int line = 0;
    int temp_line = 0;
    int column = 0;
    int temp_column = 0;
    int maxValue_cellMatrix = -1;

    // Verify if greater value is in the last column
    for (line = 0; line <= lenghtSeq2; line++)
    {
        if (maxValue_cellMatrix < SimilarityMt[line][lenghtSeq1])
        {
            maxValue_cellMatrix = SimilarityMt[line][lenghtSeq1];
            temp_line = line;
            temp_column = lenghtSeq1;
        }
    }

    // Verify if greater value is in the last line
    for (column = 0; column <= lenghtSeq1; column++)
    {
        if (maxValue_cellMatrix < SimilarityMt[lenghtSeq2][column])
        {
             maxValue_cellMatrix = SimilarityMt[lenghtSeq2][column];
             temp_line = lenghtSeq2;
             temp_column = column;
        }
    }

    // Set output value
    Matrix_cell maxValue_cell = new Matrix_cell();
    maxValue_cell.setValMatrixElement(maxValue_cellMatrix);
    maxValue_cell.setPosi(temp_line);
    maxValue_cell.setPosj(temp_column);
```

```
    return maxValue_cell;
}
```

Apêndice B

Documentação dos Sistemas Utilizados em Estudos de Caso

B.1 Elevator

A Figura B.1 ilustra o modelo de estados do sistema Elevator. Cada uma das transições numeradas no modelo estão descritas a seguir:

1. Inicia o sistema de elevadores e suas threads;
2. Usuário, fora do elevador, requisita um elevador do sistema para descer (neste caso não há nenhum elevador no andar solicitado);
3. Usuário, dentro do elevador, requisita ao elevador uma parada (neste caso o usuário deseja ir a um andar diferente do atual);
4. Usuário, fora do elevador, requisita um elevador do sistema para subir (neste caso não há nenhum elevador no andar solicitado);
5. A porta do elevador está completamente fechada;
6. Não há nenhuma requisição de subida no andar alcançado;
7. Há uma requisição de subida no andar alcanádo;
8. Existe uma requisição de parada ainda não cumprida (será a próxima a ser cumprida);
9. Não há mais requisições de parada a serem cumpridas.
10. Requisição de descida para o andar mais baixo (impossível descer a partir deste andar);
11. Requisição de descida a partir de um andar não identificado (ID incorreto);
12. Requisição de descida a partir de um andar onde já se encontra um elevador parado;

13. Requisição de parada para um andar não identificado (ID incorreto);
14. Requisição de subida a partir do último andar (impossível subir a partir deste andar);
15. Requisição de subida a partir do último andar (impossível subir a partir deste andar);
16. Requisição de subida a partir de um andar não identificado (ID incorreto);
17. Requisição de subida a partir de um andar onde já se encontra um elevador parado;
18. Requisição de um elevador a partir de um andar para subir (não há nenhum elevador parado no andar);
19. Requisição de um elevador a partir de um andar para descer (não há nenhum elevador parado no andar);
20. Requisição de parada a partir de um elevador (Elevador está num andar diferente do solicitado);
21. Encerramento do sistema de elevadores.

O alfabeto dos eventos do Elevator e a codificação de cada um dos eventos está na Tabela B.1.

B.2 Cruise Control

A seguir, na Figura B.2 encontra-se ilustrado o modelo de estados do sistema Cruise Control.

O alfabeto dos eventos do Cruise Control e a codificação de cada um dos eventos está na Tabela B.2.

B.3 WTP

O alfabeto dos eventos do WTP e a codificação de cada um dos eventos está na Tabela B.3.

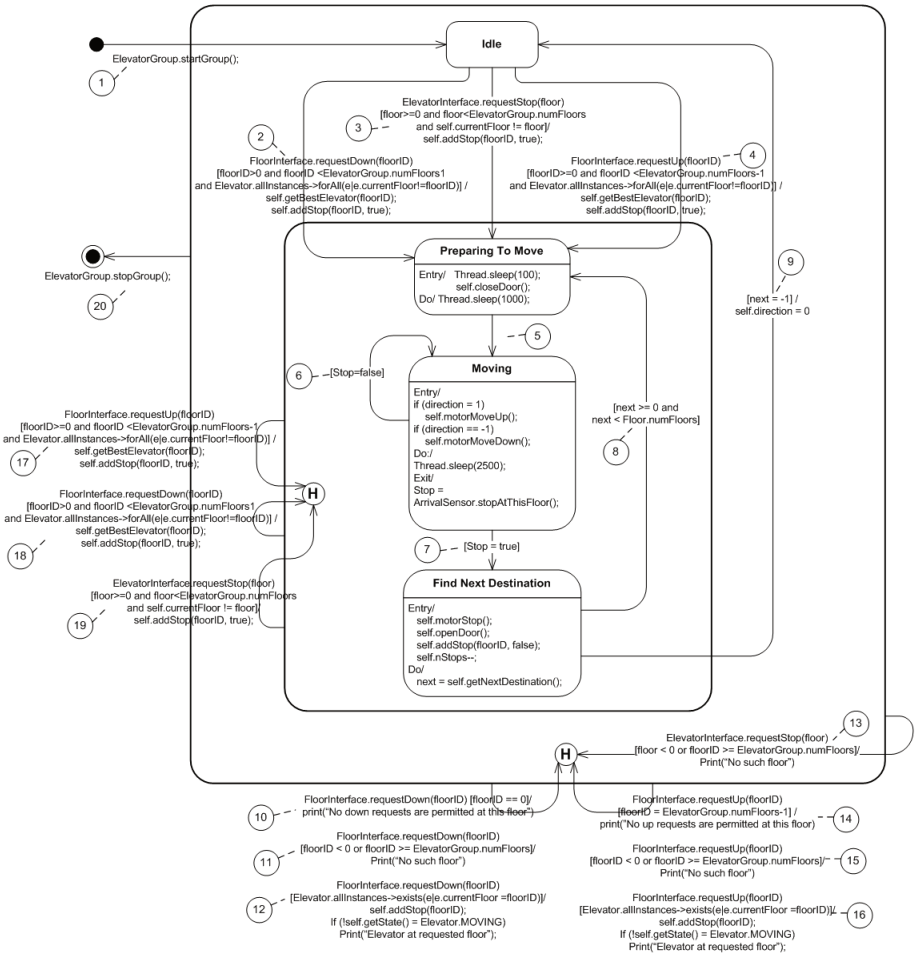


Figura B.1: Modelo de estados do estudo de caso Elevator

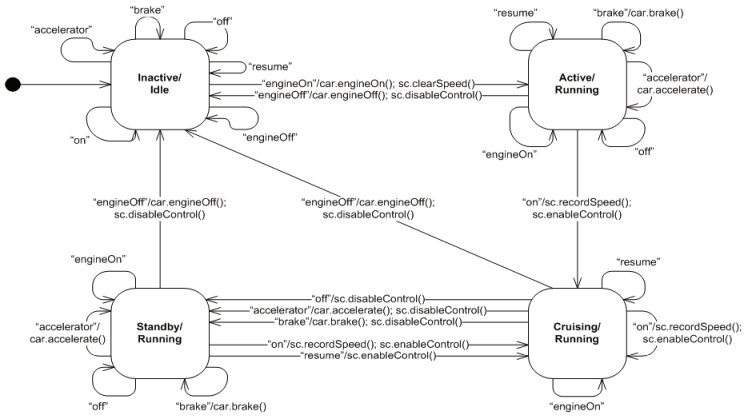


Figura B.2: Modelo de estados do estudo de caso Cruise Control

Tabela B.1: Elevator - Eventos do alfabeto Σ e respectivos códigos

Eventos	Códigos	Eventos	Códigos
Excecao	40	ElevatorGroup-ElevatorGroup	57
ElevatorGroup-startGroup	41	ElevatorGroup-getGroup	58
ElevatorGroup-stopGroup	42	ElevatorGroup-startThread	59
FloorInterface-requestUp (P)	43	Floor-getNoFloors	60
FloorInterface-requestDown (P)	44	Floor-getSensor	61
ElevatorInterface-requestStop (P)	45	Floor-removeFloors	62
Elevator-addStop (P)	46	Floor-requestDownMade	63
Elevator-openDoor (P)	47	Floor-requestUpMade	64
Elevator-closeDoor (P)	48	FloorInterface-FloorInterface	66
Elevator-getBestElevator (P)	49	FloorInterface-getFloor	67
ElevatorInterface-motorMoveUp (P)	50	FloorControl-FloorControl	65
ElevatorInterface-motorMoveDown (P)	51	ArrivalSensor-stopAtThisFloor	68
ElevatorInterface-motorStop (P)	52	Elevator-getFloor	69
FloorInterface-stopAtThisFloor (P)	53	ElevatorControl-requestStop	70
endRequest (P)	54	ElevatorGroup-motorMoving	71
ElevatorControl-getElevator	12	Floor-requestDown	72
ElevatorControl-stopElevator	14	Floor-requestUp	73
FloorInterface-requestUp	15	FloorControl-stopAtThisFloor	74
ElevatorGroup-elevatorDisplay	16	ElevatorControl-ElevatorControl	75
FloorInterface-requestDown	17	Elevator-notifyNewFloor	77
ElevatorGroup-getElevatorInterface	18	ArrivalSensor-ArrivalSensor	76
ElevatorInterface-requestStop	19	ElevatorGroup-getFloorInterface	78
ElevatorInterface-ElevatorInterface	20	Floor-Floor	79
Elevator-addStop	21	Floor-getFloorID	80
ElevatorInterface-getElevatorId	22	Floor-requestDownServiced	81
Elevator-openDoor	23	Floor-requestUpServiced	82
ElevatorControl-openDoor	24	Floor-selectFloor	83
Elevator-closeDoor	25	FloorControl-requestDown	84
ElevatorControl-closeDoor	26	FloorControl-requestUp	85
Elevator-getBestElevator	27	Elevator-Elevator	86
ElevatorInterface-getFromList	28	Elevator-getDirection	87
ElevatorInterface-motorMoveUp	29	Elevator-getDoorOpen	88
ElevatorControl-motorMoveUp	30	Elevator-getElevatorID	89
ElevatorInterface-motorMoveDown	31	Elevator-getMotorMoving	90
ElevatorControl-motorMoveDown	32	Elevator-getNextDestination	91
ElevatorInterface-motorStop	33	Elevator-getNumberOfStops	92
ElevatorControl-motorStop	34	Elevator-getState	93
FloorInterface-stopAtThisFloor	35	Elevator-getStop	94
endRequest	37	Elevator-moveElevator	95
ArrivalSensor-getTheFloor	55	Elevator-run	96
Elevator-removeAllElevators	56	Elevator-selectElevator	97
Elevator-stopElevator	98	Elevator-turnOff	99

Tabela B.2: Cruise Control - Eventos do alfabeto Σ e respectivos códigos

Eventos	Códigos
Controller-engineOn	11
Controller-engineOff	13
Controller-on	15
Controller-off	17
Controller-resume	19
Controller-brake	21
Controller-accelerator	23
CarSimulator-engineOn	25
CarSimulator-engineOff	27
SpeedControl-enableControl	29
SpeedControl-disableControl	31
SpeedControl-recordSpeed	33
SpeedControl-clearSpeed	35
CarSimulator-accelerate	37
CarSimulator-brake	39
CarSimulator-getIgnition	41
Controller-getControlState	41
Controller-getState	41
Controller-getSpeedControl	41
CruiseControl-getControl	41
SpeedControl-getState	41
SpeedControl-getCruiseSpeed	41
CarSimulator-getSpeed	41
CarSimulator-getThrottle	41
CarSimulator-getBrakePedal	41
CarSimulator-getDistance	41
CarSimulator-setThrottle	41
SpeedControl-run	45
CarSimulator-run	45
CruiseControl-handleCommand	47

Tabela B.3: WTP - Eventos do alfabeto Σ e respectivos códigos

Eventos	Códigos
RcvAbort	11
RcvAck	12
RcvErrorPDU	13
RcvInvoke	14
RcvResult	15
TimerTO_A	16
TimerTO_R	17
TimerTO_W	18
TR-Abort.req	19
TR-Invoke.res	20
TR-Invoke.req	21
TR-Result.req	22
TR-Result.res	23
Abort	24
Ack	25
CRASH	26
HANG	27
Result	28
TR-Abort.ind	29
TR-Invoke.ind	30
TR-Invoke.cnf	31
TR-Result.cnf	32
TR-Result.ind	33
ERROR	34
RcvInvoke (P)	35
TimerTO_R (P)	36
Result (P)	37

Apêndice C

Introdução à Análise com Curvas ROC

Um dos meios mais simples de avaliar os resultados de problemas de classificação binária, i.e., que apresentam somente duas classes para os resultados (positivo ou negativo) é a tabela de contingência (também chamada de matriz de confusão). Esta tabela apresenta as seguintes classificações:

- Um caso verdadeiramente positivo classificado como positivo é dito verdadeiro-positivo (VP);
- Um caso verdadeiramente negativo classificado como negativo é dito verdadeiro-negativo (VN);
- Um caso verdadeiramente positivo classificado como negativo é dito falso-negativo (FN);
- Um caso verdadeiramente negativo classificado como positivo é dito falso-positivo (FP).

Um problema que ocorre em muitos modelos de classificação é o fato de fornecerem valores contínuos como resultado. Neste caso, para obter-se a tabela de contingência é necessária a discretização dos valores existentes através da adoção de um limiar para a variável analisada.

Assim, todos os valores acima do limiar são atribuídos à uma classe e os valores abaixo do limiar são atribuídos à outra classe. Porém, para cada limiar adotado obtem-se uma tabela de contingência que pode ser diferente [84].

Obtida a tabela de contingência, uma forma de interpretar os resultados apresentados é a redução dos quatro valores na tabela (VP, VN, FN e FP) a um único valor como, por exemplo, a acurácia ou a precisão. Porém, tais medidas são influenciadas pela distribuição dos dados [34].

É possível avaliar os resultados sem que estes sejam influenciados pela distribuição. A curva ROC (*Receiver Operating Characteristics*) é um gráfico para análise da variação da sensibilidade e da especifidade em classificadores. Podemos definir sensibilidade como o número de decisões positivas de um classificador em relação a todos os casos verdadeiramente positivos e especificidade como o número de decisões negativas em relação a todos os casos verdadeiramente negativos.

Na curva ROC, ao invés de analisarmos as medidas de sensibilidade e especificade individualmente, comumente utilizamos a área abaixo da curva que associa estas duas medidas e é utilizada como medida da qualidade de um classificador [17].

Conjuntos de dados contínuos classificados considerando-se apenas um limiar específico, serão posicionados num ponto específico do gráfico ROC correspondente a $(\frac{VP}{FN+VP}, \frac{FP}{FP+VN})$.

A Figura C.1 ilustra um gráfico ROC. Nesta figura podemos destacar alguns aspectos importantes:

- O ponto $(0, 0)$ corresponde à classificação negativa de todos os dados analisados no experimento. Neste caso, o classificador obtém resultados FP ou VP;
- O ponto $(0, 1)$ corresponde à classificação positiva dos dados analisados no experimento. Neste caso, o classificador não classifica nenhum dado como FN ou VN;
- O ponto $(1, 0)$ corresponde a classificação errada de todos os dados, ou seja, não ocorrem VP e VN;
- Por fim, $(1, 1)$ representa o classificador ideal, que sempre classifica os dados de forma correta, ou seja, não ocorrem FP e FN.

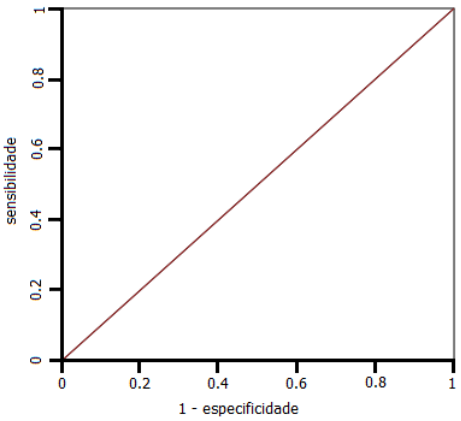


Figura C.1: Pontos da curva ROC

Quanto mais próximo do ponto $(1, 1)$ melhor é o classificador avaliado. Todo classificador que se posiciona, para um dado limiar, acima da linha diagonal obtém melhores resultados do que um classificador aleatório.

Porém, utilizando-se apenas um limiar, a avaliação do classificador fica restrita. Um problema neste caso é que alterando-se o limiar podemos aumentar a sensibilidade e diminuir a especificidade do classificador ou vice-versa [17].

De forma a analisar o classificador de uma maneira mais geral uma solução é analisá-lo independentemente de um limiar único. A análise por meio da curva ROC nos mostra, ao invés

de um único ponto, uma curva em que cada ponto representa os resultados do classificador ordenados e divididos conforme um limiar no intervalo dos possíveis valores que o limiar pode vir a ter [34]. Assim, somando-se todos os pontos obtidos a partir de cada limiar estabelecido temos a curva que indica o comportamento geral do classificador.

A área sob a curva ROC pode ser utilizada para a comparação de classificadores. Neste caso, o classificador que obtém a maior área é considerador "melhor" do que o classificador com curva de menor área.