

Universidade de São Paulo
Instituto de Matemática e Estatística
Bachalerado em Ciência da Computação

Rafael Mota Gregorut

Título da monografia
se for longo ocupa esta linha também

São Paulo
Dezembro de 2015

**Título da monografia
se for longo ocupa esta linha também**

Monografia final da disciplina
MAC0499 – Trabalho de Formatura Supervisionado.

Orientadora: Prof. Dr. Ana Cristina Vieira de Melo

São Paulo
Dezembro de 2015

Resumo

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave).

Palavras-chave: palavra-chave1, palavra-chave2, palavra-chave3.

Abstract

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa.

Keywords: keyword1, keyword2, keyword3.

Sumário

1	Introdução	1
2	Conceitos	3
2.1	Statecharts	3
2.2	Testes e critérios de teste	5
2.3	Mineração de Padrões em Sequências	6
2.4	Spefication patterns	7
2.5	Verificação formal	7
3	Tecnologias	9
3.1	Ferramenta GTSC	9
3.2	Model checker	9
4	Análise dos casos de teste e extração das propriedades	11
4.1	Análise dos critérios DS e UIO	11
4.2	Análise do critério HSwitch-Cover	11
4.3	Análise dos critérios All-simple-paths e All-paths-K-configuration	11
5	Verificação das propriedades obtidas	13
6	Conclusões	15
A	Título do apêndice	17
	Referências Bibliográficas	19

Capítulo 1

Introdução

Uma monografia deve ter um capítulo inicial que é a Introdução e um capítulo final que é a Conclusão. Entre esses dois capítulos poderá ter uma sequência de capítulos que descrevem o trabalho em detalhes. Após o capítulo de conclusão, poderá ter apêndices e ao final deverá ter as referências bibliográficas.

Para a escrita de textos em Ciência da Computação, o livro de Justin Zobel, *Writing for Computer Science* (Zobel, 2004) é uma leitura obrigatória. O livro *Metodologia de Pesquisa para Ciência da Computação* de Wazlawick (2009) também merece uma boa lida.

O uso desnecessário de termos em língua estrangeira deve ser evitado. No entanto, quando isso for necessário, os termos devem aparecer *em itálico*.

Modos de citação:

indesejável: [AF83] introduziu o algoritmo ótimo.

indesejável: (Andrew e Foster, 1983) introduziram o algoritmo ótimo.

certo : Andrew e Foster introduziram o algoritmo ótimo [AF83].

certo : Andrew e Foster introduziram o algoritmo ótimo (Andrew e Foster, 1983).

certo : Andrew e Foster (1983) introduziram o algoritmo ótimo.

Uma prática recomendável na escrita de textos é descrever as legendas das figuras e tabelas em forma auto-contida: as legendas devem ser razoavelmente completas, de modo que o leitor possa entender a figura sem ler o texto onde a figura ou tabela é citada.

Apresentar os resultados de forma simples, clara e completa é uma tarefa que requer inspiração. Nesse sentido, o livro de Tufte (2001), *The Visual Display of Quantitative Information*, serve de ajuda na criação de figuras que permitam entender e interpretar dados/resultados de forma eficiente.

Capítulo 2

Conceitos

2.1 Statecharts

A especificação de um software é um documento de referência que contém os requisitos que o programa deve satisfazer. Ela também pode ser vista como um modelo de como o software deve se comportar. É possível elaborá-la usando linguagem natural, com casos de uso, ou utilizando técnicas de engenharia formal de software.

Statecharts são um tipo de especificação formal de software baseados em máquinas de estados finitas (MEF) utilizados em sistemas complexos, como sistemas reativos (Harel *et al.*, 1987). Assim como um autômato, um statechart possui um conjunto de estados, transições e eventos de entrada que proporcionam mudanças de estados. Porém, um statechart possui os seguintes recursos adicionais:

- Ortogonalidade

Um statechart pode estar em mais de um estado simultaneamente, o que é útil para modelar situações de concorrência e paralelismo. O conjunto de estados ativos em determinado instante é denominado de configuração. Na figura 2.1, os estados F e G são ortogonais. Assim, se o statechart se encontrar no estado A e o evento a ocorrer, a configuração do statechart irá mudar para os estados F e G simultaneamente.

- Hierarquia

Um estado pode conter sub-estados e transições internas, elevando a capacidade de abstração e encapsulamento. Na figura 2.1, Os estados F e G são sub-estados do estado I . Já os estados B e C são sub-estados do estado F .

- Condições de guarda

É possível condicionar a ocorrência de uma transição ao valor de uma variável, à entrada de um estado ou à ocorrência de uma ação. Na figura 2.1, a transição do estado D para o estado E possui uma condição de guarda: $in(C)$, que é verdadeira quando o statechart se encontra no estado C e falsa caso contrário. Então, a mudança de D para E acontecerá de fato se o evento e ocorrer e a condição $in(C)$ for verdadeira, ou seja, quando e ocorrer e a configuração atual do statechart contiver o estado C .

- Broadcasting

Uma transição possui um evento de entrada, necessário para que a transição aconteça, mas também pode ter uma ação de saída, que será disparada ao final da mudança de estado. Esse artefato permite que reações em cadeia ocorram em um statechart. Na

figura 2.1, a transição do estado E para D possui um evento de entrada c e uma ação de saída d . Assim, suponha que os estados C e E constituam a atual configuração do statechart e o evento c ocorra. A transição E para D ocorrerá e desencadeará a ação d , que por sua vez acarretará a transição de C para B . No final, statechart estará com sua configuração nos estados D e B , ainda que somente o evento c tenha ocorrido.

- História

Um statechart é capaz de lembrar estados já visitados anteriormente. Na figura 2.2, o armazenamento da história do statechart é representado pelo pseudo-estado H . Se a configuração atual é D e o evento a ocorre, a próxima configuração será constituída pelo sub-estado de A mais recentemente visitado: B ou C .

Um statechart também possui um estado inicial e pode possuir um estado final. Em casos de sistemas reativos, é comum que o estado final esteja ausente. O estado inicial é representado por um círculo completamente preenchido e o estado final, por um círculo preenchido contido em uma circunferência.

Um exemplo de figura está na figura 2.1.

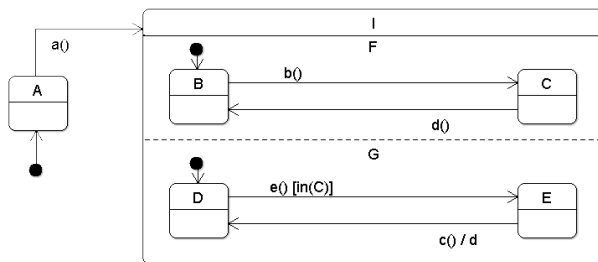


Figura 2.1: Exemplo de statechart

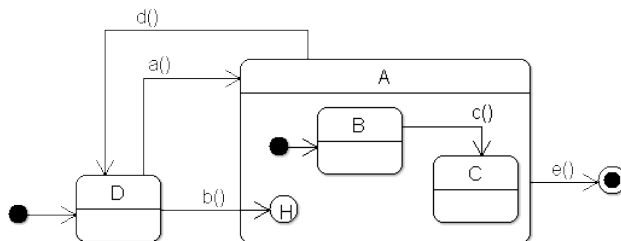


Figura 2.2: Exemplo de statechart

2.2 Testes e critérios de teste

Testar é o processo de executar um programa com o objetivo de encontrar erros (Myers *et al.*, 2012). Muitas vezes o processo de teste é visto erroneamente somente como um meio de garantir que um programa faz aquilo que deveria fazer. Entretanto, a atividade de testar uma aplicação deve ser encarada como uma forma de encontrar comportamentos e resultados inesperados. Sendo assim, pode-se dizer que um teste bem sucedido é aquele que encontra um defeito no sistema (Myers *et al.*, 2012), embora tal terminologia seja comumente utilizada na indústria para referir-se a testes que não detectem nenhuma anomalia.

As técnicas de testes podem ser classificadas em duas categorias:

- Técnica funcional (caixa-preta)

Considera-se as funcionalidades descritas na especificação para a criação dos casos de teste. É necessário estudar os comportamentos e funcionalidades especificados, criar os casos de teste correspondentes, submeter o sistema aos casos criados e analisar os resultados comparando-os com a descrição da especificação. Alguns exemplos de critérios dessa técnica são (Maldonado *et al.*, 2004):

- Particionamento em classes de equivalência

Os dados de entrada são particionados em classes válidas e inválidas de acordo com as condições presentes na especificação. Os casos de teste são criados com base em cada classe selecionando-se um elemento de cada classe como representante da classe toda. Com esse critério, os casos de teste podem ser executados de maneira sistemática de acordo com os requisitos.

- Análise do valor limite

Semelhante ao critério anterior, porém a seleção do representante de cada classe é feita na fronteira entre as classes. Desse modo, as condições associadas a valores limites são exercitadas mais rigorosamente.

Como a maioria das especificações é feita de modo descritivo, os casos de teste gerados pela técnica funcional podem ser informais e imprecisos, o que dificulta a automação de sua execução e torna necessária a intervenção humana. Entretanto, essa técnica exige somente que sejam identificados os requisitos, as entradas e saídas correspondentes, o que a torna praticável durante todas as fases de teste (unidade, integração e sistema).

- Técnica Estrutural (caixa-branca)

Considera-se a estrutura da implementação do código para a criação dos casos de teste. Nessa técnica, o programa é representado por um grafo orientado de fluxo de controle, onde cada vértice corresponde a um bloco de código, um comando por exemplo, e cada arco corresponde à transição entre os blocos. São critérios dessa técnica:

- Todos-nós

Exige que todos os vértices, ou seja, todos os comandos, sejam exercitados.

- Todos-arcos

Exige que todos os arcos, ou seja, todas as mudanças de fluxo, sejam exercitados

- Todos-caminhos

Exige que todos os caminhos possíveis do programa sejam exercitados

Os testes derivados da técnica estrutural contribuem especialmente para a manutenção, depuração e aumento da confiabilidade da implementação do programa. Entretanto, eles não representam um meio de validar o sistema, pois não levam em consideração os requisitos especificados.

Portanto, percebe-se que as duas técnicas devem ser usadas de maneira complementar. Os testes funcionais colaboram para a validações dos requisitos com base na especificação e os testes estruturais são relevantes para analisar a cobertura e confiabilidade do código escrito.

2.3 Mineração de Padrões em Sequências

Mineração de padrões em sequências é um técnica importante na área de mineração de dados e pode ser aplicada em diversas áreas, como marketing e análise de DNA. O problema de mineração de padrões em sequências pode ser definido da seguinte forma (Pei *et al.*, 2004):

Seja $I = i_1, i_2, \dots, i_n$ o conjunto de todos os itens. Um elemento é um conjunto de itens denotado por (x_1, \dots, x_m) , onde $x_k \in I$. Um item pode ocorrer no máximo uma vez em cada elemento, mas pode ocorrer em mais de um elemento de uma sequência. Uma sequência $\alpha = \langle a_1, a_2, \dots, a_n \rangle$ é subsequência de uma sequência $\beta = \langle b_1, b_2, \dots, b_m \rangle$ se existem inteiros $1 \leq j_1 < j_2 < \dots < j_n \leq m$ tal que $a_1 \subseteq b_{j_1}, \dots, a_n \subseteq b_{j_n}$. Chamamos de suporte de uma sequência α o número de tuplas em que α ocorre no banco de sequências S . Uma sequência α é dita padrão sequencial se seu suporte é maior ou igual a um mínimo arbitrário. O problema de mineração de sequências consiste em, dado um banco de sequências S e um valor mínimo de suporte, encontrar o conjunto de todos os padrões sequenciais de S .

Por exemplo:

O conjunto de itens $I = a, b, c, d, e, f$ o banco de sequências é S é dado por:

$\langle (a), (abc), (ac), (d), (cf) \rangle$
$\langle (ad), (c), (bc), (ae) \rangle$
$\langle (ef), (ab), (df), (c), (b) \rangle$
$\langle (e), (g), (af), (c), (b), (c) \rangle$

A sequência $\langle (a), (b) \rangle$ é subsequência de todas as sequências de S . Se definissemos mínimo suporte igual a 3, então um padrão obtido seria $\langle (a), (b), (c) \rangle$.

Os principais algoritmos para esse tipo de problema podem ser divididos em duas categorias:

- Algoritmos apriori: baseados na geração e teste de candidatos. Podem gerar muitos candidatos e necessitam de várias consultas ao banco. Um exemplo seria o algoritmo GSP.
- Algoritmos por crescimento de padrões: Fazem a projeção do banco de sequências em bancos menores baseando-se nos padrões encontrados até o momento para então fazer a mineração recursivamente nas projeções. Um exemplo seria o algoritmo FreeSpan. Para reduzir o tamanho das projeções do banco e a quantidade de acessos, pode-se ordenar os itens dentro de cada elemento e realizar as projeções baseando-se no prefixos dos padrões. O PReflexSpan, por exemplo, utiliza esta técnica e foi utilizado neste trabalho.

Por exemplo, o banco de sequências S mencionado anteriormente possui a seguinte projeção em relação ao prefixo $\langle (a) \rangle$:

$\langle (abc), (ac), (d), (cf) \rangle$
$\langle (d), (c), (bc), (ae) \rangle$
$\langle (b), (df), (c), (b) \rangle$
$\langle (f), (c), (b), (c) \rangle$

De acordo com a análise feita em (Pei *et al.*, 2004), o PrefixSpan é mais rápido do que o GSP e FreeSpan. Além disso, as projeções do banco geradas durante sua execução são sempre menores do que o banco original, pois contém somente sufixos das sequências frequentes.

2.4 Specification patterns

Specification patterns, ou padrões de especificação, são modelos de propriedades formais pré-definidos, que podem ser usados e adaptados de acordo com os requisitos a serem verificados. Os padrões foram definidos a partir de uma pesquisa de requisitos na literatura e em diversos projetos, somando 555 especificações coletadas de 35 fontes diferentes (Dwyer *et al.*, 1999).

Os padrões de especificação podem ser classificados em duas categorias principais: ocorrência e ordem. Os padrões de ocorrência estabelecem a ocorrência ou ausência de um determinado evento ou estado em uma região definida da especificação. Eles ainda podem ser classificados em universalidade, ausência, existência e existência limitada. Os padrões de ordem relacionam-se com a ordem de ocorrência de eventos ou estados. Eles podem ser sub-divididos em precedência, resposta, precedência em cadeia e resposta em cadeia.

Para cada padrão é necessário um escopo, o qual define a região da especificação onde a propriedade deve ser válida. Os escopos disponíveis são: global, antes de Q , depois de Q , entre Q e R e depois de Q até R , onde Q e R são eventos ou estados.

Significado de cada padrão (Dwyer *et al.*, 1998):

- Ausência: um dado estado ou evento não ocorre dentro do escopo
- Existência: um dado estado ou evento deve ocorrer dentro do escopo
- Existência limitada: um dado estado ou evento deve ocorrer k vezes dentro do escopo
- Universalidade: um dado estado ou evento ocorre em todo o escopo
- Precedência: um estado ou evento P deve sempre ser precedido por um estado ou evento Q dentro do escopo
- Resposta: um estado ou evento P deve sempre ser seguido por um estado ou evento Q dentro do escopo
- Precedência em cadeia: uma sequência de estados ou eventos P_1, \dots, P_n deve sempre ser precedida por uma sequência de estados ou eventos Q_1, \dots, Q_n
- Resposta em cadeia: uma sequência de estados ou eventos P_1, \dots, P_n deve sempre ser seguida por uma sequência de estados ou eventos Q_1, \dots, Q_n

Os padrões estão disponíveis, por exemplo, em LTL e CTL no site do projeto <http://patterns.projects.cis.ksu.edu>

2.5 Verificação formal

Capítulo 3

Tecnologias

Embora neste exemplo tenhamos apenas um capítulo, entre a introdução e a conclusão de uma monografia podemos ter uma sequência de capítulos descrevendo o trabalho e os resultados. Estes podem descrever fundamentos, trabalhos relacionados, método/modelo/algoritmo proposto, experimentos realizados, resultados obtidos.

Cada capítulo pode ser organizado em seções, que por sua vez pode conter subseções.

Um exemplo de figura está na figura 5.1.

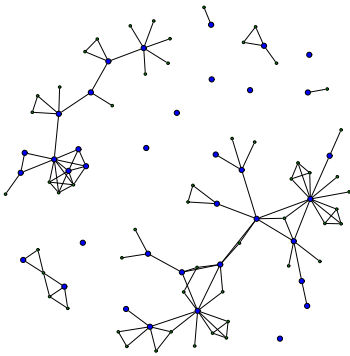


Figura 3.1: *Exemplo de uma figura.*

3.1 Ferramenta GTSC

3.2 Model checker

Capítulo 4

Análise dos casos de teste e extração das propriedades

Embora neste exemplo tenhamos apenas um capítulo, entre a introdução e a conclusão de uma monografia podemos ter uma sequência de capítulos descrevendo o trabalho e os resultados. Estes podem descrever fundamentos, trabalhos relacionados, método/modelo/algoritmo proposto, experimentos realizados, resultados obtidos.

Cada capítulo pode ser organizado em seções, que por sua vez pode conter subseções.

Um exemplo de figura está na figura 5.1.

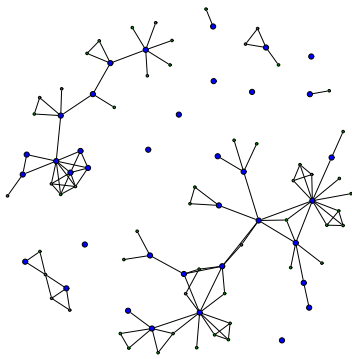


Figura 4.1: *Exemplo de uma figura.*

4.1 Análise dos critérios DS e UIO

4.2 Análise do critério HSwitch-Cover

4.3 Análise dos critérios All-simple-paths e All-paths-K-configuration

Capítulo 5

Verificação das propriedades obtidas

Embora neste exemplo tenhamos apenas um capítulo, entre a introdução e a conclusão de uma monografia podemos ter uma sequência de capítulos descrevendo o trabalho e os resultados. Estes podem descrever fundamentos, trabalhos relacionados, método/modelo/algoritmo proposto, experimentos realizados, resultados obtidos.

Cada capítulo pode ser organizado em seções, que por sua vez pode conter subseções.

Um exemplo de figura está na figura 5.1.

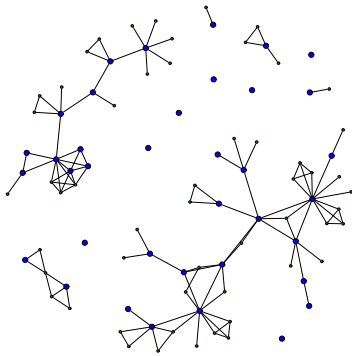


Figura 5.1: *Exemplo de uma figura.*

Capítulo 6

Conclusões

[illegible]

¹Exemplo de referência para página Web: www.vision.ime.usp.br/~jmena/stuff/tese-exemplo

Apêndice A

Título do apêndice

[illegible]

Referências Bibliográficas

- Dwyer et al.(1998)** Matthew B. Dwyer, George S. Avrunin e James C. Corbett. Property specification patterns for finite-state verification. Em *2nd Workshop on Formal Methods in Software Practice*. Citado na pág. [7](#)
- Dwyer et al.(1999)** Matthew B. Dwyer, George S. Avrunin e James C. Corbett. Patterns in property specifications for finite-state verification. Em *Proceedings of the 21st International Conference on Software Engineering*. Citado na pág. [7](#)
- Harel et al.(1987)** David Harel, Amir Pnueli, Jeanette Schmidt e Rivi Sherman. On the formal semantics of statecharts. Em *Proceedings of Symposium on Logic in Computer Science*, páginas 55–64. Citado na pág. [3](#)
- Maldonado et al.(2004)** José Carlos Maldonado, Ellen Franciane Barbosa, Auri M. R. Vincenzi, Márcio Eduardo Delamaro, Simone do Roccio Senger de Souza e Mario Jino. Introdução ao teste de software, 2004. Citado na pág. [5](#)
- Myers et al.(2012)** Glenford J. Myers, Tom Badgett e Corey Sandler. *The art of software testing*. John Wiley & Sons, Inc, 3º edição. Citado na pág. [5](#)
- Pei et al.(2004)** Jian Pei, Behzad Mortazavi-Asl e Umeshwar Dayal. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16. Citado na pág. [6](#), [7](#)
- Tufte(2001)** Edward Tufte. *The Visual Display of Quantitative Information*. Graphics Pr, 2nd edição. Citado na pág. [1](#)
- Wazlawick(2009)** Raul S. Wazlawick. *Metodologia de Pesquisa em Ciencia da Computação*. Campus, primeira edição. Citado na pág. [1](#)
- Zobel(2004)** Justin Zobel. *Writing for Computer Science: The art of effective communication*. Springer, segunda edição. Citado na pág. [1](#)