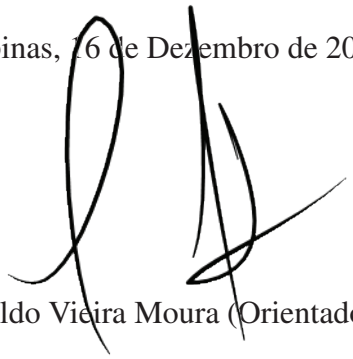
Geração de Conjuntos de Teste para Sistemas Reativos, de Tempo-Real, e com Transformações de Contexto

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Adilson Luiz Bonifácio e aprovada pela Banca Examinadora.

Campinas, 16 de Dezembro de 2009.



Arnaldo Vieira Moura (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Crisllene Queiroz Custódio – CRB8 / 7966

Bonifácio, Adilson Luiz

Sigla Geração de conjuntos de teste para sistemas reativos, de tempo-real, e com transformações de contexto / Adilson Luiz Bonifácio -- Campinas, [S.P. : s.n.], 2009.

Orientador : Arnaldo Vieira Moura

Tese (Doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Sistemas reativos. 2. Sistemas de tempo real. 3. Teste baseado em modelos. 4. Software - Testes. I. Moura, Arnaldo Vieira. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Generating test suites for reactive and real-time systems, with context transformations

Palavras-chave em inglês (Keywords): 1. Reactive systems. 2. Real-time systems. 3. Model-based testing. 4. Software - Testing.

Área de concentração: Teoria da computação e Teste de sistemas

Titulação: Doutor em Ciência da Computação

Banca examinadora: Prof. Dr. Arnaldo Vieira Moura (IC-Unicamp)
Profa. Dra. Ana Cristina Vieira de Melo (IME-USP)
Prof. Dr. Jose M. Parente de Oliveira (DCC-ITA)
Profa. Dra. Eliane Martins (IC-Unicamp)
Prof. Dr. Luiz Eduardo Buzato (IC-Unicamp)

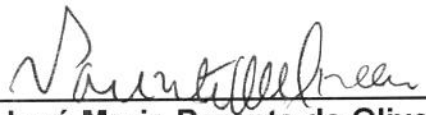
Data da defesa: 16/12/2009

Programa de Pós-Graduação: Doutorado em Ciência da Computação

TERMO DE APROVAÇÃO

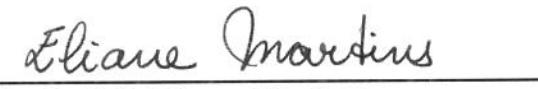
Tese Defendida e Aprovada em 16 de dezembro de 2009, pela Banca examinadora composta pelos Professores Doutores:

Profª. Drª. Ana Cristina Vieira de Melo
IME / USP

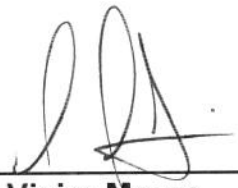


Prof. Dr. José Maria Parente de Oliveira
DCC / ITA

Prof. Dr. Luiz Eduardo Buzato
IC / UNICAMP



Profª. Drª. Eliane Martins
IC / UNICAMP



Prof. Dr. Arnaldo Vieira Moura
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Geração de Conjuntos de Teste para Sistemas Reativos, de Tempo-Real, e com Transformações de Contexto

Adilson Luiz Bonifácio[1]

Dezembro de 2009

Banca Examinadora:

- Arnaldo Vieira Moura (Orientador)
- Ana Cristina Vieira de Melo
  Departamento de Ciência da Computação - IME - USP
- Jose M. Parente de Oliveira
  Departamento de Ciência da Computação - ITA
- Eliane Martins
  Instituto de Computação - UNICAMP
- Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP
- Adenilso da Silva Simão (Suplente)
  Instituto de Ciências Matemáticas e de Computação - USP
- Ricardo de Oliveira Anido (Suplente)
  Instituto de Computação - UNICAMP

[1]Suporte financeiro de: Bolsa do CNPq (processo 141978/2008-2) 2008–2009.

Resumo

O objetivo deste trabalho é prover métodos eficientes de geração de casos de teste para sistemas reativos críticos. Sistemas dessa natureza compreendem sistemas de tempo real e com transformações de contexto. Uma das técnicas mais usadas na geração de conjuntos de teste tem sido a abordagem baseada em modelos formais. Neste caso, os formalismos fornecem uma base sólida para que a atividade de teste seja efetuada de forma precisa e segura.

Este trabalho propõe a construção de modelos formais, métodos e técnicas, bem como estratégias de teste, para dar suporte ao processo de geração automática de conjuntos de teste, aplicáveis a sistemas complexos. Porém, o processo de geração de testes baseado em modelos se torna, muitas vezes, impraticável em aplicações reais, devido ao problema da explosão combinatória de estados. Daí a necessidade de se encontrar modelos adequados que capturem o comportamento desejado dos sistemas a serem testados, bem como a importância de se construir métodos que contornem o problema da explosão do espaço de estado, de maneira razoável, permitindo que a geração de testes seja um processo aplicável a sistemas complexos.

Entre os modelos abordados neste trabalho estão: (i) as tradicionais Máquinas de Estados Finito (FSM); (ii) uma extensão das FSM usando variáveis de contexto, as Máquinas de Estados Finito Estendida (EFSM); (iii) a extensão temporizada de EFSM (TEFSM), que possui, não apenas variáveis de contexto, mas também variáveis relógio; (iv) os modelos temporizados com entradas e saídas independentes, conhecidos como *Timed I/O Automata* (TIOA); e (v) uma extensão proposta para TIOA, denominado *Timed I/O Context Automata* (TIOCA), para compreender a evolução contínua de tempo e também as transformações de contexto.

Com relação a geração de testes baseada em tais modelos, foi proposto, primeiramente, uma técnica de derivação de sequências de confirmação para TEFSM, usando *model-checking*. Em seguida, foi proposta uma generalização para um método de geração de conjuntos completos de teste usando FSM. Também foi desenvolvido um novo método de discretização do modelo TIOA, provendo a base necessária para a geração de casos de teste usando os conceitos de proposta de teste e produto síncrono. Por fim, foi desenvolvida uma extensão do método de discretização para TIOA também proposto neste trabalho, aplicado ao modelo TIOCA, permitindo a geração de testes em sistemas com evolução contínua de tempo e fluxo de dados, usando os conceitos de proposta de teste e produto de TIOCA.

Abstract

This work aims to provide efficient test case generation methods for reactive and critical systems. In general, reactive and critical systems are real-time systems with context transformations. One of the most promising techniques for generating test suites is model-based testing. The formalisms supply the basis to perform a precise and dependable testing activity. In this scenery, our work proposes a construction of formal models, methods and techniques, as well as testing strategies, to support the process of automatically generating test suites for complex systems. However, the test generation process using formal models is usually infeasible in real applications, due to the state space explosion. Therefore, we need to find out suitable models to capture the system behaviors, and also to construct methods that can overcome the explosion problem, in a reasonable way, allowing the generation of test suites for complex systems.

In this work we treat the following formal models: the conventional FSM; an extension of FSM using context variables (EFSM); the proposed extension of EFSM (TEFSM) to capture context variables and also clock variables; timed models, with disassociated input and output actions, called TIOA; and the proposed extension for TIOA, so-called TIOCA, to capture continuous time evolution and context transformations.

In a first step of this work we proposed a technique to derive confirming sequences for TEFSM, using *model-checking*. Next, a classical method to generate complete test suites was generalized for FSM. We also proposed a new discretization method for TIOA models, allowing the test case generation using test purpose and the synchronous product. Lastly, we extended the discretization method for TIOA to obtain more compact grid automata for TIOCA models, allowing the test case generation for systems with continuous time evolution and data flow transformations, using the notion of test purpose and the product of TIOCA.

“Everything is vague to a degree you do not realize
until you have tried to make it precise.”
Bertrand Russell

À Fernanda, por seu amor, apoio e compreensão ao longo desta jornada.
Aos meus pais, pelo carinho e participação em todas as minhas conquistas.

Agradecimentos

Eu gostaria de agradecer a minha esposa por sempre ter me apoiado nas horas difíceis, aos meus pais pelas oportunidades concedidas para que eu pudesse trilhar novos desafios, e a toda minha família pela força e incentivo ao longo dessa difícil trajetória.

Agradeço principalmente ao Prof. Arnaldo, meu orientador, pela ajuda, seriedade e dedicação durante o desenvolvimento deste trabalho.

Agradeço aos meus amigos pelo companheirismo e pela ótima convivência durante todo este período.

Ao CNPq pelo suporte financeiro que possibilitou o desenvolvimento dessa pesquisa.

Aos professores, funcionários e colegas do Instituto de Computação, que de alguma forma contribuíram para o desenvolvimento deste trabalho.

Aos colegas do Departamento de Computação da UEL pelo suporte e colaboração durante o período de realização deste trabalho.

Sumário

Resumo **vii**

Abstract **ix**

Agradecimentos **xv**

1 Introdução **1**
1.1 Relato Cronológico 3

2 Generalização do Método W **7**
1 Introduction 9
2 Related Works 10
2.1 Finite State Machines 10
2.2 FSM-based testing 11
3 Transition Covers 12
4 Equivalences and Stratified Families 14
5 A m-complete Test Suite 17
6 Characterization Sets 19
7 The W-method as a Particular Case 21
8 The Generalized Test Generation Method 24
9 An Example 25
10 Concluding Remarks 26
References 27

3 Discretização dos TIOA para a Verificação de Comportamentos **33**
1 Introduction 35
2 The TIOA model 37
2.1 Timed words 37
2.2 Clocks 38
2.3 Timed Input Output Automata 38

3 TIOA Discretization . . . 41
3.1 Bounded values and functions . . . 42
3.2 Grid automata . . . 46
4 Generating test cases based on test purposes . . . 58
4.1 Test purposes . . . 58
4.2 The synchronous product . . . 60
4.3 Test case generation and the grid automaton . . . 60
4.4 Applying timed test cases . . . 65
5 Related works . . . 67
6 Concluding Remarks . . . 68
References . . . 69

4 Derivando Sequências de Confirmação 75
1 Introduction . . . 77
2 Basic Formal Concepts . . . 79
2.1 Extended FSM Model . . . 79
2.2 Extended Timed Transition Systems . . . 81
3 The Timed EFSM model . . . 83
3.1 Creating a TEFSM model from an ETTS and an EFSM model . . . 83
3.2 The Operational Semantics for TEFSM models . . . 84
3.3 Configuration Distinguishability in the TEFSM model . . . 86
4 Timed Extended FSM Product . . . 86
5 Test Generation . . . 89
5.1 Conformance Testing . . . 90
5.2 Configuration Confirming Sequences . . . 90
5.3 Model-checking . . . 91
6 An Example . . . 92
7 Concluding Remarks . . . 96
References . . . 96

5 Geração de Testes para Sistemas de Tempo e Contexto 103
1 Introduction . . . 105
2 Bounds and adjusted values . . . 106
2.1 Conditions and interpretations . . . 106
2.2 Bounded values and functions . . . 107
2.3 Adjusted values . . . 109
3 Timed I/O Context Automata . . . 109
3.1 An example . . . 109
3.2 The extended model . . . 111

4 TIOCA discretization 115
4.1 Clock and context variable valuations 115
4.2 Grid automata and TIOCA 116
4.3 Relationship between a TIOCA and its grid 118
5 Generating test cases 119
5.1 Test case generation and the grid automaton 121
5.2 Applying test cases 122
6 Related works 123
7 Concluding Remarks 124
References 125

6 Conclusões **131**

Bibliografia **133**

Lista de Tabelas

Capítulo 2

Capítulo 3

1 Invariants for the product 61

Capítulo 4

Capítulo 5

Lista de Figuras

Capítulo 2

1 Specification M. 23
2 Implementation candidate M'. 24
3 Machine specification M. 26
4 A especificação M_n. 30
5 A implementação M'_n. 31

Capítulo 3

1 A BTDA model of a simple switch. 39
2 A TIOA model for a variation of the simple switch. 49
3 The partial grid automaton for Figure 2 (part 1 of 2). 50
4 The partial grid automaton for the TIOA in Figure 2 (part 2 of 2). 57
5 An example of test purpose for the switch system. 59
6 The product of a specification and a test purpose. 62
7 The partial grid automaton for Figure 6. 63
8 The framework to extract test cases. 65
9 The framework to test implementations: faulty property. 66
10 The framework to test implementations: desired property. 66

Capítulo 4

1 The TEFSM M. 93
2 The TEFSM product of M with itself. 94

Capítulo 5

1 A TIOCA model for a multimedia protocol. 110
2 The framework to test implementations: faulty property. 122
3 The framework to test implementations: desired property. 122

Capítulo 1

Introdução

O processo de verificação e teste de sistemas reativos e de tempo real vem sendo largamente estudado e pesquisado em Ciências da Computação. Entre as abordagens mais promissoras para dar suporte a este processo está a geração automática de conjuntos de testes baseada em modelos [12, 30, 39, 40]. Com o uso de formalismos adequados, requisitos e funcionalidades de sistemas podem ser especificados através de modelos matemáticos. Por isso, vários formalismos têm sido propostos para capturar o comportamento de sistemas computacionais e, a partir destes, construir conjuntos de testes, seja para sistemas discretos, para sistemas temporizados puros, e também para sistemas contendo variáveis de contexto. Porém, uma abordagem eficiente para lidar com todos estes aspectos num mesmo formalismo permanece ainda como um desafio.

Fica claro que a maneira como um sistema é modelado, ou como as características dos sistemas são capturadas pelo modelo, impactam no processo de geração de casos de teste. Outra questão importante na atividade de teste está relacionada com a decisão de quando um sistema está suficientemente testado e determinar, findo o teste, quantas falhas não cobertas, potencialmente, ainda permanecem no sistema. Em geral, para testes baseados em modelo, o objetivo não é demonstrar a equivalência entre os modelos, já que isto é impraticável em aplicações reais. A idéia é usar a noção de conformidade [17, 19] ao invés da equivalência, quando se trata da geração de casos de teste usando modelos formais. O objetivo no teste de conformidade [36, 38] é demonstrar se uma implementação candidata possui o mesmo comportamento ditado pela especificação, seguindo algum critério [38].

Uma das linhas de trabalho desta tese focou as tradicionais Máquinas de Estados Finito (FSM[1]) [13, 24, 41]. Para este modelo foi proposta uma generalização do tradicional método W [11], gerando conjuntos completos de teste [24, 26, 32, 34]. Neste caso, os resultados alcançados foram a generalização, que evita a computação de conjuntos característicos completos para gerar casos de teste, além das provas formais para a validação do método, o que permitiu mostrar que o método tradicional W é um caso particular do método generalizado.

[1] do inglês, Finite State Machines

Com o objetivo de abordar sistemas de tempo real, avançamos para um modelo mais complexo com características dinâmicas e com restrições de tempo, chamado TIOA[2] [14, 20]. Para possibilitar a geração de casos de teste em TIOA, utilizamos a noção de autômatos *grid* [15, 16], uma forma compacta para capturar a evolução contínua de tempo de maneira discreta. Neste caso, apresentamos um método de discretização, que possibilitou a geração dos testes, usando a noção de *proposta de teste* [19, 38] e de *produto síncrono* [15]. Estes conceitos permitiram a extração de execuções dos modelos, que são a base para a verificação de comportamentos entre especificações e implementações.

Agora com o objetivo de gerar testes para sistemas ainda mais complexos, propusemos um modelo chamado TEFSM[3] [5, 6], capaz de capturar a noção de tempo e fluxo de dados, ao mesmo tempo. O método proposto para TEFSM deriva testes usando a idéia de confirmação de configurações. A extração é fundamentada na utilização de um algoritmo de *model-checking* [29], onde os contra-exemplos obtidos para propriedades de falha são extraídos como sequências de confirmação.

Como o modelo TEFSM apresentou uma alta complexidade na extração de casos de teste devido ao número muito elevado de caminhos a serem explorados pelo algoritmo de model-checking, optamos por um método que usasse a noção de discretização dos modelos. Além disso, as TEFSM possuem entradas e saídas associadas, uma herança da evolução a partir das EFSM[4] [10, 11, 13, 18, 27, 33], um modelo tradicional que captura o contexto dos sistemas. Com isso a modelagem de sistemas reativos se torna limitada, já que a captura de eventos autônomos pelo modelo, produzidos pelos sistemas, é imprescindível. Por isso, mudamos a abordagem para utilizar o modelo TIOCA[5] [4], que representa tanto a evolução contínua de tempo [14, 16] quanto as transformações no fluxo de dados [25, 31, 33, 42] dos sistemas. O modelo TIOCA é uma evolução dos TIOA, que possui uma semântica de tempo bem definida. Este fato, associado a uma definição mais clara e precisa das transformações de contexto que propusemos ao modelo, nos permitiu elaborar um método capaz de gerar casos de teste de modo eficiente para o modelo TIOCA. O método propõe uma nova técnica de discretização para este modelo, e utiliza a noção de proposta de teste e do conceito de produto síncrono para a extração dos casos de teste. Além da técnica de extração apresentada, uma estratégia de aplicação dos testes em implementações candidatas para se obter vereditos de conformidade também foi elaborada.

A tese é composta por sete capítulos. O primeiro capítulo é formado por esta breve introdução, seguido por uma seção que expõe a evolução cronológica do trabalho e contextualiza todo esforço de desenvolvimento. Neste capítulo, apresentamos a conexão entre os artigos que compõem esta tese.

[2] do inglês, Timed Input/Output Automata
[3] do inglês, Timed Extended Finite State Machines
[4] do inglês, Extended Finite State Machines
[5] do inglês, Timed Input/Output Context Automata

O Capítulo 2 propõe uma generalização do tradicional método W, visando a geração de conjuntos completos de teste sem a utilização de conjuntos caracterização. Seu conteúdo é formado pelo artigo "A Generalized Model-based Test Generation Method", apresentado na *6th IEEE International Conferences on Software Engineering and Formal Methods* (SEFM'08), realizada em novembro de 2008, na Cidade do Cabo, África do Sul. O artigo foi publicado nos anais da conferência, sob a responsabilidade do *Institute of Electrical and Electronics Engineers* (IEEE). Uma extensão deste artigo, com resultados sobre a geração de testes para famílias infinitas de FSM, pode ser encontrada no relatório técnico [8], intitulado "Exponentially more Succinct Test Suites".

O Capítulo 3 aborda a geração de casos de teste para sistemas temporizados, em particular para o modelo TIOA. Uma nova técnica de discretização foi apresentada para permitir que casos de teste sejam extraídos através da modelagem de propriedades específicas dos sistemas. Esse capítulo é composto pelo artigo intitulado "A New Discretization Method for Verifying Timed System Compatible Behaviors", submetido para o periódico *Software Testing, Verification and Reliability*, da Wiley InterScience.

No Capítulo 4 é apresentada a geração de casos de teste baseada em configurações suspeitas, voltada para o modelo TEFSM. O capítulo é formado pelo artigo intitulado "Towards Deriving Test Sequences by Model Checking", publicado na revista *Electronic Notes in Theoretical Computer Science* (ENTCS), da Elsevier Science Publishers. Uma versão preliminar, intitulada "Conformance Testing by Model Checking Timed Extended Finite State Machines", foi apresentada no *Brazilian Symposium on Formal Methods* (SBMF 2006), realizado em setembro de 2006, em Natal.

O Capítulo 5 apresenta um novo método de geração de testes através da discretização do modelo TIOCA, que comporta aspectos de tempo e contexto. O método usa a noção de proposta de teste, bem como o conceito de produto síncrono, permitindo com que propriedades específicas dos sistemas sejam modeladas. O capítulo é composto pelo artigo intitulado "Generating Test Suites for Timed Systems with Context Variables", submetido para a *IEEE International Conference on Software Testing, Verification and Validation* (ICST'10) que será realizada em Paris, na França, de 06 a 09 de Abril de 2010.

No Capítulo 6 são apresentadas algumas conclusões e considerações finais. Sugestões de trabalhos futuros também são oferecidas neste capítulo.

1.1 Relato Cronológico

Com o intuito de dar suporte e automatizar o processo de geração de conjuntos de testes para sistemas reativos críticos, iniciamos o trabalho de pesquisa a procura de métodos eficientes, baseados em modelos, que pudessem ser aplicados a sistemas dinâmicos, com restrições temporais, e também com transformações de contexto, representando o fluxo de dados desses sistemas.

Os métodos de geração de casos de teste, usando a abordagem baseada em modelos, são desenvolvidos com o objetivo de verificar a conformidade entre implementações e especificações. Neste caso, o conceito de conformidade deve seguir algum critério [38], mostrando que uma implementação candidata possui o mesmo comportamento ditado pela especificação. De maneira geral, utilizamos nesta tese, a idéia de execuções e sequências, temporizadas ou não, com transformações de dados ou não, extraídas dos modelos, para verificar se especificações e implementações possuem comportamentos compatíveis.

Num primeiro momento, o trabalho desenvolvido nesta tese focou a geração de testes usando um modelo estendido das tradicionais FSM, chamado EFSM. O modelo proposto, chamado TEFSM [5, 6], estendeu as EFSM, incorporando os aspectos de tempo e contexto ao modelo. Aspectos de tempo capturam o comportamento dinâmico de um sistema, enquanto aspectos de contexto modelam o fluxo de dados e suas transformações.

O esforço nesta direção foi propor uma técnica para confirmação de configurações numa especificação, baseada na abordagem para EFSM [31]. A idéia foi fundamentada pelo uso de algoritmos de *model-checking*. Através de uma configuração suspeita é possível que caminhos que levam o sistema a uma execução falha sejam capturados pelas sequências extraídas do produto da especificação com tal configuração suspeita. Uma configuração suspeita é uma configuração do contexto, i.e. uma propriedade, da máquina de especificação que pode representar um comportamento não desejável do sistema. Já que uma propriedade pode ser especificada por uma configuração suspeita, o produto é capaz de englobar o comportamento simultâneo de ambos os modelos. Com isso uma falha pode ser encontrada através do algoritmo de model-checking, extraindo uma execução do produto, chamada de sequência de confirmação, que representa um contra-exemplo para a propriedade de falha.

No entanto, algumas dificuldades foram encontradas, principalmente no que se refere ao comportamento de sistemas reativos em aplicações reais. Um modelo formal, para ser o mais fiel possível na captura do comportamento de um sistema, deve permitir a ocorrência de eventos autônomos em resposta a eventos externos. Essa característica permite modelagens mais realistas no que se refere ao comportamento de um sistema crítico reativo. Outro ponto está relacionado com a cobertura de falhas oferecida pelo método de geração de sequências de confirmação para TEFSM. Um número muito grande de configurações suspeitas impossibilita a representação de todas as falhas, recaindo sobre o problema de explosão combinatória de estados. Para contornar esse problema, reconsideramos o passo anterior, o qual usava um modelo complexo compreendendo tanto aspectos de tempo quanto de contexto.

Em face a essas dificuldades, o trabalho foi redirecionado para, primeiramente, atacar o problema de geração de casos de teste visando um modelo mais simples, as tradicionais FSM [13, 22, 23, 24, 35, 41]. Neste caso optamos por uma cobertura completa de falhas, desenvolvendo um método que pode gerar um conjunto completo de testes para o modelo em questão. Propusemos uma generalização para o tradicional método W [11], de tal forma que podemos

gerar um conjunto completo de testes [24, 26, 32, 34] sem o uso obrigatório de um conjunto caracterização [7, 9]. Mostramos que é possível obter conjuntos completos de teste para sistemas reativos modelados através de FSM somente com o uso de subconjuntos de qualquer conjunto caracterização. Esse método evita a computação de conjuntos caracterização, bem como a computação dos índices das especificações. Além do impacto que o método causa na computação dos conjuntos de teste, mostramos também como refinar o método W a partir dessa generalização, provando que o primeiro é na verdade um caso particular do método proposto.

Seguindo a idéia de um aumento gradual na complexidade dos modelos utilizados, prosseguimos passando para um novo modelo, chamado de TIOA [20, 14], que lida com especificações temporizadas e restrições de tempo. O TIOA é capaz de especificar sistemas temporizados e suas execuções, permitindo que a evolução contínua de tempo seja representada nos modelos. O desafio neste caso foi ainda maior ao lidar com o problema de explosão de estados gerado pela captura da evolução contínua de tempo. Para contornar tal problema, utilizamos a noção de discretização com o objetivo de diminuir o número de estados representados no modelo final e discretizado. Uma discretização clássica sobre os conhecidos *timed automata* resultam nos chamados autômatos *grid* [15, 16], que se utilizam das tradicionais regiões de relógio [2, 28]. No entanto, nossa proposta se diferencia da discretização clássica baseada em regiões de relógio, pois permitimos que a granularidade do grid seja escolhida de forma mais flexível. Essa flexibilidade proporciona escolhas de granularidades que diminuem o número de estados gerados no modelo discretizado. Provamos que mesmo um grid sendo obtido através de granularidades mais flexíveis, os modelos originais de TIOA podem simular homomorficamente seus correspondentes grid, e vice-versa. Essa flexibilidade pode resultar na construção de autômatos grid menores, já que não temos a obrigação de manter uma relação rígida entre a granularidade escolhida e o número de variáveis relógio contidas no modelo original.

Este resultado forneceu a base para a geração de testes a partir do formalismo de TIOA. Optamos pela noção de proposta de teste [19, 38] para que propriedades específicas dos sistemas fossem modeladas. Propostas de teste e especificações de sistemas, modeladas por TIOA, aliadas a outro importante conceito, o de produto, permitem a modelagem do comportamento coordenado de ambos, e permitem a extração de execuções tanto dos sistemas quanto das propostas de teste. Assim, de um autômato grid, obtido do produto, são extraídas as execuções que representam o comportamento coordenado, e por consequência resultam em execuções que levam o sistema às propriedades de falha, ou às propriedades desejadas conforme o caso. Dessa forma, é possível extrair os conjuntos de teste para propriedades específicas, e então usar o conceito de conformidade para verificar se implementações candidatas seguem os comportamentos modelados pela especificação e pela proposta de teste [40]. Adotamos neste trabalho [3] uma estratégia de realimentação de sequências, baseada na idéia de subsequências de execuções [21] para verificar a compatibilidade de comportamentos.

Com a experiência obtida do trabalho anterior, e o resultado de uma nova técnica de dis-

cretização para TIOA, seguimos com o objetivo de gerar conjuntos de testes para um modelo ainda mais complexo, que possa expressar tanto a evolução contínua de tempo [14, 16] quanto as transformações no fluxo de dados [25, 31, 33, 42]. Nessa nova vertente propusemos um modelo para englobar ambos os aspectos. Mais do que isso, esse novo modelo permite a independência dos eventos de entrada e saída, diferente do modelo TEFSM, onde as entradas e saídas estão intimamente associadas. De posse do novo modelo, denominado TIOCA, foi necessário desenvolver uma nova discretização para este formalismo [4, 15], inspirada na idéia de discretização proposta para os modelos TIOA [3]. Novamente desviando da noção clássica de regiões de relógio, os autômatos grid obtidos permitem a extração de conjuntos de teste dos modelos discretizados. Lembrando que no modelo TIOCA lidamos tanto com variáveis relógio quanto com as variáveis de contexto e suas transformações.

Preparando para a geração dos casos de teste, mostramos novamente que o modelo TIOCA [4] homomorficamente simula seus respectivos autômatos grid, e vice-versa. Com o objetivo de especificar propriedades específicas dos sistemas, retomamos a noção de proposta de teste, agora para o modelo TIOCA. O conceito de produto síncrono para TIOCA também foi definido de forma que o comportamento conjunto de uma especificação e de uma proposta de teste possa ser capturado. A partir do método de discretização, da utilização da proposta de teste e também do conceito de produto síncrono, mostramos como extrair automaticamente conjuntos de teste. A estratégia de realimentação para TIOA foi adaptada para ser aplicada ao modelo TIOCA, permitindo que vereditos de conformidade possam ser obtidos para implementações candidatas [21, 40].

Com isso fechamos o objetivo principal proposto na tese, de elaborar um método para geração de conjuntos de testes baseado em modelos, para sistemas críticos e de tempo real, com transformações de contexto.

Capítulo 2

Generalização do Método W

Prólogo

O teste de conformidade tem como objetivo demonstrar se o comportamento de uma implementação conforma com o comportamento da especificação de um sistema. Na geração de casos de teste vários modelos formais podem ser usados como formalismo base. Neste trabalho usamos o conhecido modelo FSM, largamente estudado na literatura.

Este trabalho apresenta uma generalização do método W, que usa a noção de conjunto caracterização para a geração de conjuntos completos de casos de teste baseado no modelo FSM. Um conjunto completo de teste garante uma cobertura completa de falhas no sistema modelado. O novo algoritmo dessa generalização, denominado método G, é capaz de gerar conjuntos completos de teste da mesma forma que o método W, porém na ausência de um conjunto caracterização completo. Nesse sentido, o método proposto gera tais conjuntos usando apenas subconjuntos de qualquer conjunto caracterização. Mostramos também que o método W na verdade é um caso particular do método generalizado G.

Este capítulo é composto pelo artigo, "A Generalized Model-based Test Generation Method", que generaliza o método clássico W. O objetivo é mostrar que o método proposto é capaz de gerar conjuntos de testes sem a presença de um conjunto caracterização. O artigo foi publicado nos anais do *6th IEEE International Conferences on Software Engineering and Formal Methods* (SEFM'08), realizado na Cidade do Cabo, África do Sul, de 10 a 14 de Novembro de 2008.

A Generalized Model-based Test Generation Method

Adilson Luiz Bonifácio* Arnaldo Vieira Moura†
Computing Institute, University of Campinas
P.O. 6176 – Campinas – Brazil – 13081-970
adilson@ic.unicamp.br arnaldo@ic.unicamp.br

Adenilso da Silva Simão
Mathematic Science and Computing Institute, University of São Paulo
P.O. 668 – São Carlos – Brazil – 13560-970
adenilso@icmc.usp.br

Abstract

In this paper we present a generalization to the W-method [3], which can be used for automatically generating test cases. In contrast to the W-method, this generalization allows for test case generation even in the absence of characterization sets for the specification. We give proofs of correctness for this generalization, and show how to derive the original W-method from it as a particular case. Proofs of correctness for the W-method, not given in the original paper, are also presented in a clear and detailed way.

1 Introduction

Conformance testing aims at demonstrating that the implementation behavior conforms to the behavior dictated by the specification [7, 20, 21]. In the literature, there are many model-based test derivation methods for conformance testing of critical and reactive systems [4, 14, 22]. The problem of generating test cases for conformance testing has been intensively studied, specially for models based on Finite State Machines (FSMs) [5, 9, 10, 11, 19, 23]. One of the most well-known of these test generation methods is the W-method [3], which uses the notion of characterization sets. The W-method was proposed for deterministic FSMs and it has been widely investigated, and many variations have been developed around its main ideas [11, 12, 13, 18].

*Supported by CNPq grant 141978/2008-2
†Supported by CNPq grant 472504/2007-0

In this paper we present a generalization of the W-method. This generalization allows us to derive a m-complete test suite without using a characterization set. A test suite is m-complete if it guarantees a complete fault coverage[17], while considering deterministic FSM implementations with up to m states. In fact, our method can generate test suites using only subsets of any characterization set. We discuss how to refine the generalization in order to arrive at the original W-method, demonstrating that the latter is a particular case of our method. Proofs of correctness are presented in a clear form, including the correctness for the original W-method.

This paper is organized as follows. In Section 2 we describe some work related to our proposal, and we review some basic concepts. The concept of transition covers for FSMs is presented in Section 3. In Section 4 we introduce equivalence in FSMs, and stratified families of sets. The generation of a complete test suite is presented in Section 5. In Section 6 we reconsider characterization sets. How to refine our method in order to obtain the original W-method is described in Section 7. In Section 8 we present the algorithm for the generalized method, and illustrate its usefulness with an example in Section 9. Finally, in Section 10, we give some concluding remarks.

2 Related Works

This section reviews the FSM model and some important related notions. We also present more details about the W-method and other variant model-based test generation methods, such as the W_p and HSI methods.

2.1 Finite State Machines

The basic model used to capture a system behavior is the FSM. Formally, a FSM [8] is a system $M = (X, Y, S, s_0, \delta, \lambda)$ given by:

- a finite input alphabet, X;
- a finite output alphabet, Y;
- a finite set of states, S;
- an initial state $s_0 \in S$; and
- output and transition functions, respectively, $\lambda : X \times S \to Y$ and $\delta : X \times S \to S$.

Note that such a machine is deterministic and complete. A FSM is called complete if for each state s of M, there is a transition from s with input symbol a, for every $a \in X$. A deterministic FSM does not allow two different transitions going out of the same state with identical input symbols.

Successive applications of the transition function δ give rise to the extended transition function $\widehat{\delta} : X^\star \times S \to S$, defined by

$$\begin{aligned} \widehat{\delta}(\epsilon, s) &= s, \\ \widehat{\delta}(a\rho, s) &= \widehat{\delta}(\rho, \delta(a, s)), \text{where } a \in X \text{ and } \rho \in X^\star. \end{aligned}$$

Here, ϵ will denote the empty word. For convenience, if $\widehat{\delta}(\rho, s_1) = s_2$ we also write $s_1 \xrightarrow{\rho} s_2$.

We extend λ to $\widehat{\lambda} : X^\star \times S \to Y^\star$ thus

$$\begin{aligned} \widehat{\lambda}(\epsilon, s) &= \epsilon, \\ \widehat{\lambda}(a\rho, s) &= \lambda(a, s)\widehat{\lambda}(\rho, \delta(a, s)), \text{with } a \in X,\ \rho \in X^\star. \end{aligned}$$

Henceforth, unless mention to the contrary, we will assume that M and M' denote FSMs in the form $M = (X, Y, S, s_0, \delta, \lambda)$ and $M' = (X, Y', S', s'_0, \delta', \lambda')$. Note that M and M' have the same input alphabet.

The reachability notion expresses the idea of starting at the initial state, traversing some transitions, and reaching a target state.

Definition 1 *A state s in a FSM M is reachable if and only if there exists $\rho \in X^\star$ such that $\widehat{\delta}(\rho, s_0) = s$.* •

We also say that $\widehat{\lambda}(\rho, s)$ is the behavior of M from state s over the input sequence ρ. The behavior of M over ρ is simply the behavior of M from s_0 over ρ. A sequence ρ distinguishes two states s_1 and s_2 of M if ρ gives distinct behaviors for s_1 and s_2, that is, if $\widehat{\lambda}(\rho, s_1) \neq \widehat{\lambda}(\rho, s_2)$.

2.2 FSM-based testing

Here, we briefly describe the basic W-method, which can be used for test generation using FSM models. We also briefly describe the related W_p and the HSI methods.

The W-method The objective is to verify whether implementation models conform to a specification model, as characterized by the behavior responses generated by external stimuli [3].

Basically, the application of this method consists in two main steps, given a specification FSM M and an implementation FSM M': (i) test sequences generation, based on M; and (ii) application of each test sequence to M and M', followed by a comparison of their respective behaviors.

The technique uses characterization sets of M in order to obtain a complete set of test case sequences. A characterization set, loosely speaking, can distinguish every pair of machine states (see Section 6). Let W be a characterization set for M. In order to obtain the test sequences, the W-method prefixes the sequences in W with certain sequences from $X^\star$, thus obtaining a set Z containing extended sequences. Furthermore, the method also computes a cover set P for M. Basically, applying sequences from P one can traverse any edge of M. The desired set of test sequences is the product PZ. More details to be presented in the sequel.

The W_p-method A related technique, the so called W_p-method [6], can potentially reduce the total length of the test sequences generated by the basic W-method. Again, let W be a characterization set for the specification model, M. For each state s_i of M, an identification subset $W_i \subseteq W$ is obtained. The idea is that for each state s_j of M, with $s_i \neq s_j$, there exists an input sequence $\rho_j \in W_i$ such that s_i and s_j are distinguishable by ρ_j, and no other proper subset of W_i has this property.

Then, a checking sequence for each state is prefixed to all sequences in the corresponding identification set. A checking sequence for a given state is simply an input sequence reaching that state, when starting at the initial state. It is proven [6] that the length of the resulting test sequences may be shorter, compared to those sequences obtained using the complete PZ concatenation of the basic W-method.

The HSI-method The HSI-method [16] uses the notion of trace-inclusion and a quasi-equivalence relation to verify conformance between partial non-deterministic FSM implementations and a given FSM specification. For that, so called harmonized sate identification sets are used instead of the identification subsets used in the W_p-method. Whereas identification sets fixed the sequences associated with a specific state s_i, a harmonized state identification set D_i, is constructed by taking prefixes of a characterization set W, but now allowing the reuse of a same prefix for different states. Distinguishing sequences for states are then taken from the intersection of D_i-sets. The discussion in [16] affirms that shorter sequences can be found to distinguish every pair of states in M.

3 Transition Covers

Let M be a FSM. A cover set $P \subseteq X^\star$ is required to exercise every transition in M, *i.e.*, for every transition $\delta(a, s) = r$ in M there must be $\rho, \rho a \in P$ such that $\widehat{\delta}(\rho, s_0) = s$. In this way, we can obtain a behavior of M that reaches state s, and terminates by traversing the specific edge from s to r, labeled by a.

The cover set notion is formalized next.

Definition 2 *A set of input sequences $C \subseteq X^\star$ is a cover set for a FSM M if for every pair of states $s, r \in S$ and every input symbol $a \in X$, with $\delta(a, s) = r$, there exist $\rho, \rho a \in C$ such that $\widehat{\delta}(\rho, s_0) = s$.* •

A cover set can be obtained by constructing a labeled tree for M. A labeled tree is a system $T = (N, A, l_v, l_e)$, where N is a set of nodes, A is the set of edges, and $l_v : N \rightarrow S$ and $l_e : A \rightarrow X$ are labeling functions of nodes and edges, respectively. The nodes in the tree will be labeled by states of M and edges will be labeled by symbols from X.

Construction 3 *A labeled tree for M, $T = (N, A, l_v, l_e)$, can be constructed as follows:*

1. *Initiate with $N = \{n_0\}$, $A = \emptyset$, $l_v(n_0) = s_0$ and $l_e = \emptyset$, where s_0 is the initial state of M and n_0 is the root of T. We say that n_0 has level zero in T.*
2. *Inductively, suppose T is already constructed up to level $k \geq 0$. Level $k+1$ is constructed by inspecting of nodes in level k from left to right:*
   (a) *let $n \in N$ be the next node to be inspected.*
   (b) *if there already exists $m \in N$ with $l_v(m) = l_v(n)$, and m is at some level $l < k$ in T, then node n is ignored, and we take the next node at level k. Otherwise, for every input $a \in X$ and every $r \in S$ with $r = \delta(a, l_v(n))$, we add a new node n' to N, a new edge (n, n') to A, and define $l_v(n') = r$ and $l_e(n, n') = a$. We then proceed to the next node in level k.*
3. *Step 2 is repeated if new nodes were added to T in the last iteration; otherwise, T is completed.* •

The process will always terminate since the set of states in M is finite. Depending on how the symbols from X are selected, different trees can be obtained (see step 2b in Construction 3).

The next definition shows how to construct a required cover set.

Definition 4 *Let T be a labeled tree for M. The set P_T is defined by all words $\alpha \in X^\star$ which label paths in T, starting at the root.* •

Note that $\epsilon \in P_T$. When T is clear from the context, we will use the simplified notation P instead of P_T.

We can now show that P_T, from Definition 4, is a cover set for machine M. Before that, we need a property of labelled trees.

Lemma 5 *Let $T = (N, A, l_v, l_e)$ be a labeled tree for a FSM M, as given by Construction 3. Let P_T be the set obtained as in Definition 4. Let $\rho \in X^\star$ and $s \in S$ be such that $\widehat{\delta}(\rho, s_0) = s$. Then, there exists a node $n \in N$ with $l_v(n) = s$. Furthermore, there exists a sequence $\alpha \in P_T$ with $\widehat{\delta}(\alpha, s_0) = s$ and such that for every edge $\delta(a, s) = r$ we have $\alpha a \in P_T$.*

Proof Directly from Construction 3. Details in [2]. •

Now we can enunciate the cover set property.

Corollary 6 *Let $T = (N, A, l_v, l_e)$ be the labeled tree for a FSM M, as given by Construction 3. Let P_T be the set obtained as in Definition 4. If every state of M is reachable, then the set $P_T \subseteq X^\star$ is a cover set for M.*

Proof Let $\delta(a, r) = s$ be an edge. As r is reachable, we have $\widehat{\delta}(\rho, s_0) = r$, for some $\rho \in X^\star$. By Lemma 5, we have α, $\alpha a \in P_T$ with $\widehat{\delta}(\alpha, s_0) = r$, for some $\alpha \in X^\star$. Thus, P_T is a cover set for M. •

4 Equivalences and Stratified Families

This section deals with state equivalence relations induced by the transition functions of the extended machines. The next definition exposes those notions in a general context.

Definition 7 *Let M and M' be two FSMs over the same input alphabet, X, and let s and s' be states of M and M', respectively.*

1. *Let $\rho \in X^\star$. We say that s is ρ-equivalent to s' if $\widehat{\lambda}(\rho, s) = \widehat{\lambda}'(\rho, s')$. In this case, we write $s \approx_\rho s'$. Otherwise, s and s' are ρ-distinguishable and we write $s \not\approx_\rho s'$.*

2. *Let $K \subseteq X^\star$. We say that s is K-equivalent to s' if s is ρ-equivalent to s', for every $\rho \in K$. In this case, we write $s \approx_K s'$. Otherwise, s and s' are K-distinguishable and we write $s \not\approx_K s'$.*

3. *Let $k \geq 0$. We say that s is k-equivalent to s' if s is X^k-equivalent to s'. Otherwise, s and s' are k-distinguishable. We write, respectively, $s \approx_k s'$ and $s \not\approx_k s'$.*

4. *State s is equivalent to s' if s is k-equivalent to s', for every $k \geq 0$. Otherwise, s and s' are distinguishable. We write, respectively, $s \approx s'$ and $s \not\approx s'$.* •

We will avoid overloading the notation by indicating M and M' explicitly, *e.g.*, in the form $\approx_k^{M,M'}$, since both machines will always be clear from the context. Definition 7, obviously, also applies when M and M' are the same machine. In this case, it is easy to verify that all relations defined above are, in fact, equivalence relations over the state set of the machine. Hence, each such equivalence relation $\approx_Z$ gives rise to a partition $[Z]$ of the state set S.

Definition 8 *Let M be a FSM. The index of M, ι_M, is the number of equivalence classes induced by the $\approx$ relation over the states of M.* •

Clearly, we will always have $1 \leq \iota_M \leq |S|$, where S is the state set of M.

The next lemma gathers some simple observations.

Lemma 9 *Let M and M' be two FSMs with states s and s', respectively.*

1. *Let $K \subseteq X^\star$. If $s \approx_K s'$, then $s \approx_L s'$, for every L with $L \subseteq K$. On the other hand, if $s \not\approx_K s'$, then $s \not\approx_L s'$, for every L with $K \subseteq L$.*

2. *Let $k \geq 0$. If $s \approx_k s'$ then $s \approx_l s'$ for every l with $l \leq k$. On the other hand, if $s \not\approx_k s'$, then $s \not\approx_l s'$, for every l with $l \geq k$.*

3. *Let $K, L \subseteq X^\star$. If $s \not\approx_K s'$, then $s \not\approx_{KL} s'$, for every $L \neq \emptyset$.*

Proof Trivial. •

In the sequel, we will be considering specific sets of input sequences.

Definition 10 *Let $Z_i \subseteq X^\star$, $i \geq 0$, where X is an alphabet. We say that $\{Z_i\}_{i\geq 0}$ is a stratified family over X if*

1. *$Z_0 \neq \emptyset$; and*

2. *$(X \cup \{\epsilon\})Z_i = Z_{i+1}$, for every $i \geq 0$.* •

It is easy to see that these properties are independent of each other.

Another characterization for stratification is given as follows.

Proposition 11 *Let $Z_i \subseteq X^\star$, $i \geq 0$, where X is an alphabet and with $Z_0 \neq \emptyset$. Then, the family $\{Z_i\}_{i\geq 0}$ is stratified if and only if $Z_k = \bigcup_{j=0}^{k} X^j Z_0$ for every $k \geq 0$.*

Proof Details can be found in [2]. •

The next result guarantees that certain sequences always have continuations in some of the Z_k sets.

Lemma 12 *Let $\{Z_i\}_{i\geq 0}$ be a stratified family over X and let $k \geq 0$. Then*

1. *$Z_k \subseteq Z_j$, for every $j \geq k$; and*

2. *For every $\alpha \in X^j$, with $0 \leq j \leq k$, there exists $\beta \in X^\star$ such that $\alpha\beta \in Z_k$.*

Proof From Proposition 11, we deduce that $Z_i \subseteq Z_{i+1}$, for every $i \geq 0$. A simple induction establishes item (1). For item (2), since $Z_0 \neq \emptyset$, we take $\gamma \in Z_0$. Since $j \leq k$, we take $\sigma \in X^{k-j}$. Hence, $\alpha\sigma\gamma \in X^k Z_0$. From Proposition 11 we conclude $\alpha\sigma\gamma \in Z^k$. •

Let M be a FSM and let $Z \subseteq X^\star$ be a set of input sequences. We indicate by $[Z]$ the partition induced by Z (see observation after Definition 7) over the states of M, i.e, $s \approx_Z r$ if and only if $s, r \in w$, for some $w \in [Z]$. Let $[Z_1]$ and $[Z_2]$ be two partitions over S. Then we say that $[Z_2]$ *refines* $[Z_1]$ if and only if for all $w_2 \in [Z_2]$ there exists some $w_1 \in [Z_1]$ such that $w_2 \subseteq w_1$.

The next result expresses properties of these partitions.

Lemma 13 *Let $\{Z_i\}_{i\geq 0}$ be a stratified family over the alphabet X of a FSM M. Then*

1. *$[Z_{i+1}]$ refines $[Z_i]$, for every $i \geq 0$; and*
2. *if $|[Z_k]| = |[Z_{k+1}]|$ for some $k \geq 0$, then we must have $[Z_k] = [Z_{k+1}] = [Z_{k+2}]$.*

Proof We show each item, in turn.

For item (1), assume that it does not hold for some $i \geq 0$. Then we will have states s and r such that $s \approx_{Z_{i+1}} r$ and $s \not\approx_{Z_i} r$. From Lemma 9(1) and Lemma 12(1) we deduce $s \not\approx_{Z_{i+1}} r$, a contradiction.

Now we verify item (2). From item (1), we know that $[Z_{k+1}]$ refines $[Z_k]$. Then $[Z_k] = [Z_{k+1}]$, otherwise we would have $|[Z_k]| < |[Z_{k+1}]|$. Again continuing by contradiction, assume that $[Z_{k+1}] \neq [Z_{k+2}]$. Since $[Z_{k+2}]$ refines $[Z_{k+1}]$, we will have states r and s such that $s \not\approx_{Z_{k+2}} r$ and $s \approx_{Z_{k+1}} r$. Hence, we obtain $\rho \in Z_{k+2}$, with $\rho = a\beta$ and $a \in X$, and such that $s \not\approx_{a\beta} r$. We also conclude that $a\beta \notin Z_{k+1}$, otherwise we would have the contradiction $s \not\approx_{Z_{k+1}} r$. Therefore, from Definition 10(2), we deduce $a\beta \in XZ_{k+1}$, and so, $\beta \in Z_{k+1}$.

Let s_1, $r_1 \in S$ with $s_1 = \delta(a, s)$, $r_1 = \delta(a, r)$. If $s_1 \not\approx_{Z_{k+1}} r_1$ then $s_1 \not\approx_{Z_k} r_1$, because we already know that $[Z_k] = [Z_{k+1}]$. Hence, we would have $\gamma \in Z_k$ with $\widehat{\lambda}(\gamma, s_1) \neq \widehat{\lambda}(\gamma, r_1)$. From Definition 10(1) we have $XZ_k \subseteq Z_{k+1}$, and then $a\gamma \in Z_{k+1}$. But,

$$\begin{aligned} \widehat{\lambda}(a\gamma, s) &= \lambda(a, s)\widehat{\lambda}(\gamma, s_1) \\ \widehat{\lambda}(a\gamma, r) &= \lambda(a, r)\widehat{\lambda}(\gamma, r_1). \end{aligned}$$

Then we have $\widehat{\lambda}(a\gamma, s) \neq \widehat{\lambda}(a\gamma, r)$, thus forcing the contradiction $s \not\approx_{Z_{k+1}} r$. We conclude that $s_1 \approx_{Z_{k+1}} r_1$.

Since $\beta \in Z_{k+1}$, we deduce $s_1 \approx_\beta r_1$. Again,

$$\begin{aligned} \widehat{\lambda}(a\beta, s) &= \lambda(a, s)\widehat{\lambda}(\beta, s_1) \\ \widehat{\lambda}(a\beta, r) &= \lambda(a, r)\widehat{\lambda}(\beta, r_1), \end{aligned}$$

and, since we already have $\widehat{\lambda}(\rho, s) \neq \widehat{\lambda}(\rho, r)$, we conclude that $\lambda(a, s) \neq \lambda(a, r)$. From $a \in X$ and Lemma 12(2) we infer $\sigma \in X^\star$ with $a\sigma \in Z_{k+1}$. Hence, we have $s \not\approx_{a\sigma} r$, contradicting $s \approx_{Z_{k+1}} r$. •

The next result gives the equality of successive partitions.

Corollary 14 *Let $\{Z_i\}_{i\geq 0}$ be a stratified family over the input alphabet X of a FSM M. If $|[Z_k]| = |[Z_{k+1}]|$ for some $k \geq 0$, then $[Z_k] = [Z_{k+l}]$ for every $l \geq 0$.*

Proof When $l = 0$, the result is immediate. When $l = 1$ or $l = 2$, the result follows directly from Lemma 13(2). Assume the result holds for every j, $0 \leq j \leq l$, with $l \geq 2$. We want to show that the result holds for $l + 1$. From the induction, we have $[Z_k] = [Z_{k+l}]$ and $[Z_k] = [Z_{k+l-1}]$. Hence, $[Z_{k+l-1}] = [Z_{k+l}]$. Using Lemma 13(2), we obtain $[Z_{k+l-1}] = [Z_{k+l}] = [Z_{k+l+1}]$. Hence, $[Z_k] = [Z_{k+l+1}]$, as required. •

Now let M be a FSM with m states. Suppose we have a stratified family for X, $\{Z_i\}_{i\geq 0}$, in which Z_0 partitions the states of M in $n \leq m$ equivalence classes. We want to study the partitions over states of M induced by the Z_i sets, for $i \geq 0$. The next lemma establishes the basic result.

Lemma 15 *Let M be a FSM with index m. Let $\{Z_i\}_{i\geq 0}$ be a stratified family for X such that Z_0 partitions the states of M in at least $n \leq m$ equivalence classes. Then $|[Z_i]| \geq n + i$, for every i, with $0 \leq i \leq m - n$.*

Proof When $i = 0$ we have $n + i = n$ and, from the hypothesis, $|[Z_0]| \geq n$, establishing the base. Assume the result for every j, $0 \leq j \leq i$, with $i < m - n$. We are going to show that the result holds for $i + 1$. If $|[Z_i]| \geq n + i + 1$ then $|[Z_{i+1}]| \geq n + i + 1$ (from Lemma 13(1)), and the induction is extended in this case.

Now, let $|[Z_i]| < n + i + 1$. From the induction hypothesis we conclude that $|[Z_i]| = n + i$. Since $m \geq n + i + 1$ is the index of M, there exist nonequivalent states in M, r and s, with $r \approx_{Z_i} s$. Then, $s \not\approx_{X^k} r$, for some $k \geq 0$ (see Definition 7). From Lemma 12(2), we conclude $s \not\approx_{Z_k} r$. If $k \leq i$, Lemma 12(1) would force $Z_k \subseteq Z_i$. Using Lemma 9(1) we would have $s \not\approx_{Z_i} r$, a contradiction. Hence, $k > i$.

If $|[Z_i]| = |[Z_{i+1}]|$ then, by Corollary 14, we get $Z_i = Z_k$, forcing again the contradiction $s \not\approx_{Z_i} r$. Since $[Z_{i+1}]$ refines $[Z_i]$, we can not have $|[Z_{i+1}]| < |[Z_i]|$. We conclude that $|[Z_{i+1}]| > |[Z_i]|$. But, since $|[Z_i]| = n + i$, we deduce the result desired, that is, $|[Z_{i+1}]| \geq n + i + 1$. •

Using this result, it will be easy to confirm that some $Z \in \{Z_i\}_{i\geq 0}$ will distinguish every pair of nonequivalent states.

Corollary 16 *Let M be a FSM with index m. Let $\{Z_i\}_{i\geq 0}$ be a stratified family for X such that Z_0 partitions the states of M in at least $n \leq m$ equivalence classes. Then Z_{m-n} will distinguish every pair of nonequivalent states of M.*

Proof From Lemma 15, it follows that $|[Z_{m-n}]| \geq n + (m - n) = m$. Since $[Z_{m-n}]$ is the partition induced by Z_{m-n}, we conclude that Z_{m-n} partitions states of M in m classes. Since M has index m, we conclude that Z_{m-n} will distinguish every pair of nonequivalent states of M. •

5 A m-complete Test Suite

Let M and M' be two FSMs operating over the same alphabet X. Machine M represents a specification and M' represents a possible implementation. We want to obtain a set $K \subseteq X^\star$ such that $s_0 \not\approx s'_0$ if and only if $s_0 \not\approx_K s'_0$. Such a set K is a m-complete test suite, where m

is an upper bound on the index of M'. Given K, if we want to test whether M and M' have distinct behaviors, it is enough to apply the sequences in K to both machines and compare the corresponding output sequences.

We obtain the required set by combining a cover set for M with a stratified family for M'. The next lemma establishes an auxiliary result.

Lemma 17 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m and that P is a cover set for M. Let $Z \subseteq X^\star$ be nonempty and such that Z partitions the states of M' in at least m equivalence classes. If $s_0 \approx_{PZ} s'_0$ and $s_0 \not\approx s'_0$, then there exist $\gamma \in X^\star$, $s \in S$, $s' \in S'$ such that $\widehat{\delta}(\gamma, s_0) = s$, $\widehat{\delta'}(\gamma, s'_0) = s'$ and $s \not\approx_Z s'$.*

Proof This proof can be found in [2]. •

Now we are in a position to enunciate the result which will give us the capability of testing two machines for equivalence.

Theorem 18 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m and that P is a cover set for M. Let $Z \subseteq X^\star$ be nonempty and such that Z partitions states of M' in at least m equivalence classes. Then, $s_0 \approx s'_0$ if and only if $s_0 \approx_{PZ} s'_0$.*

Proof If $s_0 \approx s'_0$ then, trivially, $s_0 \approx_{PZ} s'_0$.

For the opposite direction, assume $s_0 \approx_{PZ} s'_0$. For the sake of contradiction, assume $s_0 \not\approx s'_0$. From Lemma 17, we obtain $\beta \in X^\star$, $s \in S$ and $s' \in S'$ with $\widehat{\delta}(\beta, s_0) = s$, $\widehat{\delta'}(\beta, s'_0) = s'$, and $s \not\approx_Z s'$. We can assume, without loss of generality, that $|\beta|$ is minimal. If $\beta = \epsilon$, we would have $s = s_0$ and $s' = s'_0$, and then $s_0 \not\approx_Z s'_0$. But, since $\epsilon \in P$, this would force the contradiction $s_0 \not\approx_{PZ} s'_0$. We conclude that $\beta = \alpha a$, with $a \in X$. Let $r \in S$ and $r' \in S'$ with $\widehat{\delta}(\alpha, s_0) = r$, $\widehat{\delta'}(\alpha, s'_0) = r'$, $\delta(a, r) = s$ and $\delta'(a, r') = s'$. Using the minimality of $|\beta|$ we have $r \approx_Z r'$.

On the other hand, since P is a cover set for M, from the edge $\delta(a, r) = s$ we obtain $\rho \in P$ and $\rho a \in P$ with $\widehat{\delta}(\rho, s_0) = r$. Let $r'' \in S'$ with $\widehat{\delta'}(\rho, s'_0) = r''$. If we had $r \not\approx_Z r''$, we would obtain $\gamma \in Z$ with $\widehat{\lambda}(\gamma, r) \neq \widehat{\lambda'}(\gamma, r'')$. But then

$$\begin{aligned} \widehat{\lambda}(\rho\gamma, s_0) &= \widehat{\lambda}(\rho, s_0)\widehat{\lambda}(\gamma, r) \quad \text{and} \\ \widehat{\lambda'}(\rho\gamma, s'_0) &= \widehat{\lambda'}(\rho, s'_0)\widehat{\lambda'}(\gamma, r''). \end{aligned}$$

Hence, $\widehat{\lambda}(\rho\gamma, s_0) \neq \widehat{\lambda'}(\rho\gamma, s'_0)$, giving the contradiction $s_0 \not\approx_{\rho\gamma} s'_0$ with $\rho\gamma \in PZ$. We conclude that $r \approx_Z r''$.

Since we already have $r \approx_Z r'$, we obtain $r' \approx_Z r''$. Since Z partitions the states of M' in m classes and m is the index of M', we conclude that $r' \approx r''$. Now, from $s \not\approx_Z s'$, we obtain

$\sigma \in Z$ with $\widehat{\lambda}(\sigma, s) \neq \widehat{\lambda'}(\sigma, s')$. But,

$$\begin{aligned}
\widehat{\lambda}(\rho a\sigma, s_0) &= \widehat{\lambda}(\rho, s_0)\lambda(a, r)\widehat{\lambda}(\sigma, s) \quad \text{and} \\
\widehat{\lambda'}(\rho a\sigma, s'_0) &= \widehat{\lambda'}(\rho, s'_0)\widehat{\lambda'}(a\sigma, r'') \\
&= \widehat{\lambda'}(\rho, s'_0)\widehat{\lambda'}(a\sigma, r') \\
&= \widehat{\lambda'}(\rho, s'_0)\lambda'(a, r')\widehat{\lambda'}(\sigma, s').
\end{aligned}$$

Then, $\widehat{\lambda}(\rho a\sigma, s_0) \neq \widehat{\lambda'}(\rho a\sigma, s'_0)$. But $\rho a\sigma \in PZ$ and we would have $s_0 \not\approx_{PZ} s'_0$, contradicting the hypothesis. This concludes the proof. •

Combining the previous results, we have the following corollary, useful to determine whether two FSMs have distinguishing behaviors.

Corollary 19 *Let M and M' two FSMs operating over the same input alphabet, X. Assume that M' has index m. Assume also that P is a cover set for M, that $R \subseteq X^\star$ is nonempty and that it partitions the states of M' in at least $n \leq m$ equivalence classes. Then, s_0 and s'_0 are equivalent if and only if s_0 and s'_0 are PZ-equivalent, where $Z = \bigcup_{i=0}^{m-n} X^i R$.*

Proof Let $Z_k = \bigcup_{i=0}^{k} X^i R$, $k \geq 0$. From Proposition 11 we have that such family $\{Z_k\}_{k\geq 0}$ is stratified. From Corollary 16 we conclude that Z distinguishes every pair of nonequivalent states of M'. Then the result follows directly from Theorem 18. •

6 Characterization Sets

From the previous corollary, it might appear that Z and M are independent, since the only hypothesis involving M, in that corollary, is that P is a cover set for M. But, in fact, there is a relationship between Z and M. Before we expose the relationship between Z and M, we need another auxiliary result.

Lemma 20 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that all states of M are reachable and that $s_0 \approx s'_0$. Let $Z \subseteq X^\star$ be a set partitioning the states of M' in m equivalence classes, where m is the index of M'. Then Z distinguishes every pair of nonequivalent states of M.*

Proof Let s_1, $s_2 \in S$ with $s_1 \not\approx s_2$ and assume $s_1 \approx_Z s_2$. Since all states of M are reachable, we have $\rho_1, \rho_2 \in X^\star$ such that $\widehat{\delta}(\rho_i, s_0) = s_i$, with $i = 1, 2$. In M' we would have some $s'_1, s'_2 \in S'$ and with $\widehat{\delta'}(\rho_i, s'_0) = s'_i$, where $i = 1, 2$.

Now let $\beta \in Z$. We have,

$$\begin{aligned}
\widehat{\lambda}(\rho_2\beta, s_0) &= \widehat{\lambda}(\rho_2, s_0)\widehat{\lambda}(\beta, s_2) \\
\widehat{\lambda'}(\rho_2\beta, s'_0) &= \widehat{\lambda'}(\rho_2, s'_0)\widehat{\lambda'}(\beta, s'_2)
\end{aligned}$$

and, since $s_0 \approx s'_0$, we obtain $\widehat{\lambda}(\beta, s_2) = \widehat{\lambda}'(\beta, s'_2)$ and $\widehat{\lambda}(\rho_2, s_0) = \widehat{\lambda}'(\rho_2, s'_0)$. Since β is arbitrary, we conclude that $s_2 \approx_Z s'_2$.

Similarly,

$$\begin{aligned}\widehat{\lambda}(\rho_1\beta, s_0) &= \widehat{\lambda}(\rho_1, s_0)\widehat{\lambda}(\beta, s_1)\\ \widehat{\lambda}'(\rho_1\beta, s'_0) &= \widehat{\lambda}'(\rho_1, s'_0)\widehat{\lambda}'(\beta, s'_1),\end{aligned}$$

and we conclude that $s_1 \approx_Z s'_1$, together with $\widehat{\lambda}(\rho_1, s_0) = \widehat{\lambda}'(\rho_1, s'_0)$.

Putting it together, and knowing that $s_1 \approx_Z s_2$, we obtain $s_1 \approx_Z s'_2$ and also $s_2 \approx_Z s'_1$. Hence, $s'_1 \approx_Z s'_2$. But s'_1 and s'_2 are states of M' and so the hypothesis over Z gives $s'_1 \approx s'_2$.

On the other hand, since $s_1 \not\approx s_2$, we obtain $\sigma \in X^\star$ such that $\widehat{\lambda}(\sigma, s_1) \neq \widehat{\lambda}(\sigma, s_2)$. Now,

$$\begin{aligned}\widehat{\lambda}(\rho_1\sigma, s_0) &= \widehat{\lambda}(\rho_1, s_0)\widehat{\lambda}(\sigma, s_1)\\ \widehat{\lambda}'(\rho_1\sigma, s'_0) &= \widehat{\lambda}'(\rho_1, s'_0)\widehat{\lambda}'(\sigma, s'_1).\end{aligned}$$

Hence, from $\widehat{\lambda}(\rho_1\sigma, s_0) = \widehat{\lambda}'(\rho_1\sigma, s'_0)$ and $\widehat{\lambda}(\rho_1, s_0) = \widehat{\lambda}'(\rho_1, s'_0)$, we deduce $\widehat{\lambda}(\sigma, s_1) = \widehat{\lambda}'(\sigma, s'_1)$.

Similarly,

$$\begin{aligned}\widehat{\lambda}(\rho_2\sigma, s_0) &= \widehat{\lambda}(\rho_2, s_0)\widehat{\lambda}(\sigma, s_2)\\ \widehat{\lambda}'(\rho_2\sigma, s'_0) &= \widehat{\lambda}'(\rho_2, s'_0)\widehat{\lambda}'(\sigma, s'_2),\end{aligned}$$

and then $\widehat{\lambda}(\sigma, s_2) = \widehat{\lambda}'(\sigma, s'_2)$. However, since we already know that $s'_1 \approx s'_2$ and this leads to the contradiction $\widehat{\lambda}(\sigma, s_1) = \widehat{\lambda}(\sigma, s_2)$. This shows that the initial hypothesis was false. Hence, whenever $s_1 \not\approx s_2$ holds we must also have $s_1 \not\approx_Z s_2$, establishing the result. •

A set in these conditions is called a characterization set of M.

Definition 21 *Let M be a FSM and W a set of input sequences. W is a characterization set for M if W distinguishes any pair of nonequivalent states of M.* •

The required relation between M and Z says that Z is a characterization set of M, under certain hypothesis.

Theorem 22 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m and that P is a cover set for M. Assume also that $W \subseteq X^\star$ is nonempty and partitions the states of M' in at least $n \leq m$ equivalence classes. If $s_0 \approx_{PZ} s'_0$ then $Z = \bigcup_{i=0}^{m-n} X^i W$ is a characterization set for M.*

Proof From Proposition 11 and from Corollary 16 we conclude that Z distinguishes every pair of nonequivalent states of M'. Since P is cover set for M, we conclude that every state of M is reachable. From $s_0 \approx_{PZ} s'_0$, together with Corollary 19, we deduce $s_0 \approx s'_0$. Now we can use Lemma 20 and obtain that Z distinguishes every pair of nonequivalent states of M. From Definition 21, Z is a characterization set for M. •

It is also easy to see that the reverse does not hold. For that, let M and M' be two FSMs. It is clear that $W = X^\star$ partitions the states of M and M' in the maximum number of equivalence classes. In this case, we will have $Z = W = X^\star$ and, obviously, Z is a characterization set for M and M'. But it is not the case that we will always have $s_0 \approx s'_0$, as it is easy to construct a counter-example.

Next result shows that, under relaxed conditions, when two FSMs are equivalents both must have the same index.

Theorem 23 *Let M and M' be two FSMs operating over the same input alphabet, X. Let n and n' be the index of M and M', respectively. Assume that all states from both FSMs are reachable. If $s_0 \approx s'_0$ then $n = n'$.*

Proof For the sake of contradiction, and without loss generality, we will assume $n < n'$.

Let $s'_i \in S'$, $1 \leq i \leq n'$, be states from each one of n' equivalence classes induced by $\approx$ in S'. Since all states of M' are reachable, we obtain $\rho_i \in X^\star$ with $\widehat{\delta'}(\rho_i, s'_0) = s'_i$, $1 \leq i \leq n'$. In M, we will have some $s_i \in S$ such that $\widehat{\delta}(\rho_i, s_0) = s_i$, $1 \leq i \leq n'$. Since $n < n'$, without loss generality, we can say that $s_1 \approx s_2$.

Take any $z \in X^\star$. We have

$$\begin{aligned} \widehat{\lambda}(\rho_1 z, s_0) &= \widehat{\lambda}(\rho_1, s_0)\widehat{\lambda}(z, s_1) \quad \text{and} \\ \widehat{\lambda'}(\rho_1 z, s'_0) &= \widehat{\lambda'}(\rho_1, s'_0)\widehat{\lambda'}(z, s'_1). \end{aligned}$$

Since $s_0 \approx s'_0$, it follows that $\widehat{\lambda}(z, s_1) = \widehat{\lambda'}(z, s'_1)$. Similarly, $\widehat{\lambda}(z, s_2) = \widehat{\lambda'}(z, s'_2)$.

But since $s_1 \approx s_2$, we obtain $\widehat{\lambda}(z, s_1) = \widehat{\lambda}(z, s_2)$. Therefore, $\widehat{\lambda'}(z, s'_1) = \widehat{\lambda'}(z, s'_2)$. Since $z \in X^\star$ is arbitrary, we conclude that $s'_1 \approx s'_2$, a contradiction given that s'_1 and s'_2 are in distinct classes in M'.

Hence, we must have $n \geq n'$. Similarly, $n' \geq n$, and then $n = n'$. •

The same result indicates that when the $\approx$ relation induces a different number of equivalence classes in two FSMs, these machines can not be equivalent to each other (under the weak hypothesis of Theorem 23). On the other hand, it is simple to obtain two nonequivalent FSMs, in a such way that the $\approx$ relation induces the same number of equivalence classes in both machines. For details, see [2].

7 The W-method as a Particular Case

Consider the hypothesis of Theorem 22. We can show that W is a characterization set of M if n is the index of M and the behaviors of both machines must match.

Corollary 24 *Let M and M' be two FSMs operating over the same input alphabet, X, and assume that all states in M' are reachable. Assume further that M' has index m, that P is a cover set for M and that M has index n. Assume also that $W \subseteq X^\star$ is nonempty and partitions the states of M' in at least $n \leq m$ equivalence classes. If $s_0 \approx_{PZ} s'_0$, where $Z = \bigcup_{i=0}^{m-n} X^i W$, then $n = m$, $Z = W$ and W is a characterization set for M.*

Proof Since $s_0 \approx_{PZ} s'_0$, together with Corollary 19, we conclude that $s_0 \approx s'_0$. Next, we infer that $n = m$, from Theorem 23. Hence, $Z = W$. Therefore, by Theorem 22, W is a characterization set for M. •

When W is a characterization set for M we can guarantee the partitioning of M' in a number of classes at least equal to the index of M, if the machines are to be PZ-equivalent.

Lemma 25 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m, that M has index n and that P is a cover set for M, with $n \leq m$. Assume also that $W \subseteq X^\star$ is a characterization set for M and that $s_0 \approx_{PZ} s'_0$, where $Z = \bigcup_{i=0}^{m-n} X^i W$. Then W partitions M' in at least n equivalence classes.*

Proof We know that M has n equivalence classes: Let $C_1, \ldots, C_n$ be these classes. Let $s_i \in C_i$ and $s_j \in C_j$, where $1 \leq i < j \leq n$. Then since W is a characterization set for M, we have $s_i \not\approx_W s_j$. Since P is cover set of M, we have $\widehat{\delta}(\rho, s_0) = s_i$, for some $\rho \in P$. We also know that $\widehat{\delta}'(\rho, s'_0) = s'_i$, for some s'_i of M'. Since $s_0 \approx_{PZ} s'_0$, we get $s_i \approx_Z s'_i$. Since $W \subseteq Z$, then $s_i \approx_W s'_i$. In the same way, we have s'_j of M' with $s_j \approx_W s'_j$. Then we obtain $s'_i \not\approx_W s'_j$, otherwise $s_i \approx_W s_j$. We conclude that W partitions M' in at least $n \leq m$ equivalence classes. •

Now we can use Lemma 25 to show another version of Corollary 19, under the hypothesis that the basic set of input sequences is a characterization set for the specification.

Theorem 26 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m, that P is a cover set for M and that M has index n, with $n \leq m$. Assume also that $W \subseteq X^\star$ is a characterization set for M and that $s_0 \approx_{PZ} s'_0$, where $Z = \bigcup_{i=0}^{m-n} X^i W$. Then $s_0 \approx s'_0$.*

Proof Assume $s_0 \approx_{PZ} s'_0$. Use Lemma 25 to show that W partitions M' in at least n classes. Now use Corollary 16 to show that Z partitions M' in m classes. Finally, use Theorem 18. •

The next result is the main postulate of the basic W-method, as given in [3].

Theorem 27 *Let M and M' be two FSMs operating over the same input alphabet, X. Assume that M' has index m, that P is a cover set for M and that M has index n, with $n \leq m$. Assume also that $W \subseteq X^\star$ is a characterization set for M. Then $s_0 \approx s'_0$ if and only if $s_0 \approx_{PZ} s'_0$, where $Z = \bigcup_{i=0}^{m-n} X^i W$.*

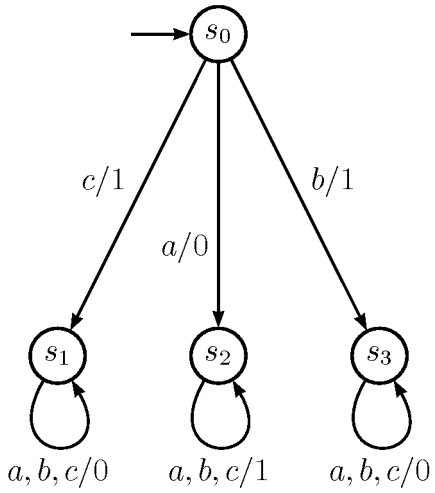


Figure 1: Specification M.

Proof If $s_0 \approx s'_0$, then $s_0 \approx_{PZ} s'_0$, trivially. For the other direction, use Theorem 26. •

In general, W need not be a characterization set for M (see Corollary 19). For the method to work, we need only guarantee that M' will be partitioned in at least n equivalence classes with $n \leq m$, where m is the index of M'. No relationship between W and M is needed. On the other hand, when using the basic W-method directly, we need to obtain a characterization set W for M, we need to know the index of M, and we also need to secure the relationship $n \leq m$. When W is not a characterization set for M, the method may fail, as shown by the following example.

Example 28 The alphabet of M and M' is $X = \{a, b, c\}$. See Figures 1 and 2. It is easy to see that M has index $n = 3$, because $s_1 \approx s_3$. The index of M' is $m = 3$ since $s'_1 \approx s'_3 \approx s'_4$. Hence $m = n$, and we would be left with $Z = W$ (see Theorem 27). Now take $W = \{\epsilon\}$. A cover set can be given by $P = \{\epsilon, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$. Then, $PZ = PW = P$. It is easy to see that M and M' are PZ-equivalent. But $s_0 \approx s'_0$ is not true. To see that, take $\alpha = bbb$. We have $\widehat{\lambda}(\alpha, s_0) = 100$ and $\widehat{\lambda'}(\alpha, s'_0) = 101$. Note how W induces only one equivalence class in M'. Therefore, clearly, W is not a characterization set for M. •

In general, it would be important to devise a mechanism by which we could obtain the number of classes induced by W in M'. First, because in this case we might avoid calculating a characterization set for M when using our more general method. Secondly, we could potentially reduce the size of the sequences in Z, when W partitions M' in k classes, with $k > n$, given that $Z = \bigcup_{i=0}^{m-n} X^i W$.

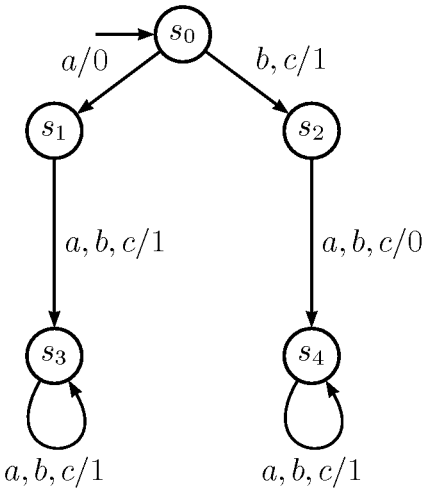


Figure 2: Implementation candidate M'.

8 The Generalized Test Generation Method

Algorithm 1 presents the generalized model-based test generation method. The input parameters are: M represents a system specification, M' is an implementation candidate for the specification M, R is any set of input sequences, n is a lower bound on the number of classes induced by R in M', and m is an upper bound on the index of M'. Thus, the method requires knowledge of a lower bound on the number n of equivalence classes induced by R in the implementation machine M', as well as an upper bound on the index m of M'. In an extreme case, once can set $n = 1$ and $m = |S'|$, that is, set m to the number of states in M'. Note that the implementation M' is given as a black box. So, we do not have access to its internal structure, and the parameters n and m must be estimated. As for the specification M, R may partition it in any number k of classes. Of course, if M and M' turn out to be equivalent, then they will have the same index and Z will, in fact, be a characterization set for both M and M'.

If the condition $n \leq m$ is secured and it turns out that M and M' are not equivalent, the algorithm produces a particular input sequence σ that is a witness to this fact, that is, M and M' display distinct behaviors over σ.

In order to apply the basic W-method (see Theorem 27) some extra effort must be applied to compute the index of M as well as a characterization set for M.

In our proposal, we do not need characterization sets, nor is it necessary to inform the index of the specification machine M. On the other hand, practical information about M can aid in obtaining a good candidate for R. For example, based on the number of symbols in the input

```
1  Input: M, M', R, m, n
2  begin
3      Obtain a cover set P for M;
4      if n ≤ m then
5          Compute Z = ⋃_{i=0}^{m−n} X^i R;
6          Compute PZ;
7      else
8          mesg: M and M' are not equivalent;
9          return;
10     end
11     foreach σ ∈ PZ do
12         Apply σ to M and to M';
13         Obtain y = λ̂(σ, s_0) and y' = λ̂'(σ, s'_0);
14         if y ≠ y' then
15             mesg: M and M' are not equivalent;
16             mesg: σ is an input witness;
17             return;
18         end
19     end
20     mesg: M and M' are equivalents;
21     return;
22 end
```

Algorithm 1: Generalized test generation algorithm.

alphabet and on the number of states and transitions in M, some distinguishing sequences can be inserted into R. Then, it is easy to obtain the set Z using the notion of stratification. Clearly, after obtaining the concatenation PZ, we can use this product to verify conformance between the specification and several proposed implementations.

Note that the size of the PZ set depends on the algorithm used to obtain the cover set P. In fact, this algorithm is polynomial in the size of M (see Section 3). Furthermore, it depends on the choice of the set R and the bound m.

9 An Example

We apply the generalized algorithm to a simple example.

Example 29 Let a specification M be given as in Figure 3. Then M has $k = 4$ states, its input alphabet is $X = \{a, b\}$, its output alphabet is $Y = \{0, 1\}$, and its transition function is as depicted in the figure.

As we can see, some transitions over the input a produce either the output 0 or the output 1. Hence, there are at least two distinct classes. Now, if we use the sequences aa and ba there is a good chance that such sequences can distinguish other states as well. Therefore, we take $R = \{aa, ba\}$ and assume that R partitions M', an implementation candidate, in at least $n = 3$ equivalence classes. If we accept $m = 5$ as a maximum on the number of states in M', we have all input conditions for Algorithm 1 secured.

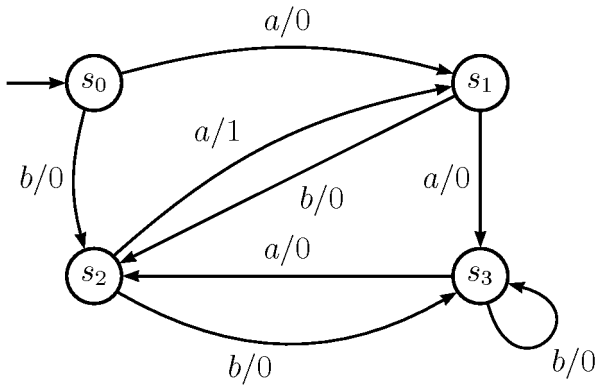


Figure 3: Machine specification M.

Note that R is not a characterization set for M because we have states s_0 and s_1 in the same equivalence class induced by R.

Next we calculate a cover set P for M. In the example, using the labeled tree construction (see Section 3) we get $P = \{\epsilon, a, b, aa, ab, ba, bb, aab, aaa\}$.

Now, with $m = 5$, $n = 3$ and $R = \{aa, ba\}$, we compute $Z = \bigcup_{i=0}^{m-n} X^i R$ and obtain

$$Z = \{aa, ba, aaa, aba, baa, bba, aaaa,$$
$$abaa, baaa, bbaa, aaba, abba, baba, bbba\}.$$

Then, the concatenation PZ will count 56 sequences.

10 Concluding Remarks

The Finite State Machine (FSM) model is well established and has been intensively investigated as a foundation for the automatic generation of test cases. The W-method is a well known technique used to compute test sequences having FSMs as its basic formal model.

In this paper, our contribution is threefold. First, we generalized the basic W-method, avoiding the computation of characteristic sets and indexes. Secondly, we demonstrated in a clear way how the basic W-method follows from our generalized method. And finally, we presented detailed proofs of correctness both of our algorithm, as well as for the main tenets of the basic W-method, the latter being absent in the original work where it was introduced.

Note that some recent test generation methods, such as those presented in Section 2.2, have to calculate a characterization set of the specification, in the same way as the basic W-method. On the other hand, our method does not need characterization sets in order to generate test cases.

We envisage that similar ideas can be used to extend and generalize other test case generation techniques, such as the W_p and HSI methods.

As future steps we plan to integrate the results presented in this paper with extensions of the basic FSM model, now also taking into account time constraints [1, 15].

References

[1] A. L. Bonifácio, A. V. Moura, A. da Silva Simão, and J. C. Maldonado. Towards deriving test sequences by model checking. *Electron. Notes Theor. Comput. Sci.*, 195:21–40, 2008.

[2] A. L. Bonifácio, A. V. Moura, and A. d. S. Simão. A generalized model-based test generation method. Technical Report IC-08-014, Instituto de Computação, Universidade Estadual de Campinas, Campinas, May 2008.

[3] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Softw. Eng.*, 4(3):178–187, 1978.

[4] S. J. Cunning and J. W. Rozenblit. Automating test generation for discrete event oriented embedded system s. *J. Intell. Robotics Syst.*, 41(2-3):87–112, 2005.

[5] R. Dorofeeva, K. El-Fakih, and N. Yevtushenko. An improved conformance testing method. In *FORTE*, pages 204–218, 2005.

[6] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Trans. Softw. Eng.*, 17(6):591–603, June 1991.

[7] A. Gargantini. Conformance testing. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*, pages 87–111. Springer-Verlag, 2005.

[8] A. Gill. *Introduction to the theory of finite-state machines*. McGraw-Hill, New York, 1962.

[9] G. Gonenc. A method for the design of fault detection experiments. *IEEE Trans. Comput.*, 19(6):551–558, 1970.

[10] F. C. Hennie. Fault detecting experiments for sequential circuits. In *FOCS*, pages 95–110, 1964.

[11] R. M. Hierons. Separating sequence overlap for automated test sequence generation. *Automated Software Engg.*, 13(2):283–301, 2006.

[12] M. Krichen. State identification. In M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*, pages 87–111. Springer-Verlag, 2005.

[13] G. Luo, G. von Bochmann, and A. Petrenko. Test selection based on communicating nondeterministic finite-state ma chines using a generalized wp-method. *IEEE Trans. Softw. Eng.*, 20(2):149–162, 1994.

[14] B. Nielsen and A. Skou. Test generation for time critical systems: Tool and case study. *ecrts*, 00:0155, 2001.

[15] A. Petrenko, S. Boroday, and R. Groz. Confirming configurations in efsm testing. *IEEE Trans. Softw. Eng.*, 30(1):29–42, 2004.

[16] A. Petrenko and G. v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. In G. Luo, editor, *IWPTS '94: 7th IFIP WG 6.1 international workshop on Protocol test systems*, pages 95–110, London, UK, UK, 1995. Chapman & Hall, Ltd.

[17] A. Petrenko and N. Yevtushenko. Testing from partial deterministic fsm specifications. *IEEE Trans. Comput.*, 54(9):1154–1165, 2005.

[18] A. Rezaki and H. Ural. Construction of checking sequences based on characterization sets. *Computer Communications*, 18(12):911–920, 1995.

[19] D. Sidhu and T. Leung. Experience with test generation for real protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 257–261, New York, NY, USA, 1988. ACM.

[20] D. P. Sidhu and T. kau Leung. Formal methods for protocol testing: A detailed study. *IEEE Trans. Softw. Eng.*, 15(4):413–426, 1989.

[21] J. Tretmans. Test generation with inputs, outputs, and quiescence. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 of *Lecture Notes in Computer Science*, pages 127–146. Springer, 1996.

[22] J. Tretmans. Testing concurrent systems: A formal approach. In J. Baeten and S. Mauw, editors, *CONCUR '99: Proceedings of the 10th International Conference on Co ncurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 46–65, London, UK, 1999. Springer-Verlag.

[23] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Trans. Comput.*, 46(1):93–99, 1997.

Epílogo

Este capítulo apresentou a generalização do clássico método W para a obtenção de um conjunto completo de casos de teste usando FSM. A generalização, chamada de método G, evita a computação de conjuntos característicos e dos índices das especificações na obtenção dos conjuntos de teste.

O objetivo foi mostrar que o método proposto gera conjuntos completos de casos de teste na ausência de conjuntos característicos das especificações. Também foi mostrado no trabalho que o método W é um caso particular do método proposto. Provas formais de tal resultado foram apresentadas de forma clara e precisa.

Um outro resultado nesta linha foi uma extensão do artigo apresentado neste capítulo, visando a geração de conjuntos de testes para famílias infinitas de FSM.

Mostramos que existem famílias infinitas de FSM tais que a aplicação do método G produz conjuntos de teste muito menores e mais compactos do que aqueles produzidos pelo método W.

Considere um alfabeto X e um conjunto de teste $V \subseteq X^\star$. O teste através do conjunto V requer a observação do comportamento dos modelos de especificação e de implementação sobre todos os elementos de V. Por isso, não é necessário considerar nenhum prefixo próprio de V, já que o teste com a maior sequência já revelaria comportamentos de qualquer prefixo menor.

Definition 30 *Seja X um alfabeto e $V \subseteq X^\star$. Defina* pff(V) *como o conjunto de elementos livre de prefixo de V, isto é,*

$$\text{pff}(V) = \{\varphi \in V \mid \varphi\vartheta \notin V \textit{ para todo } \vartheta \in X^\star \textit{ com } \vartheta \neq \varepsilon\}.$$

A eficiência de um conjunto de teste $V \subseteq X^\star$ será medida através da soma dos tamanhos de todos os elementos de pff(V).

Definition 31 *Seja X um alfabeto e $V \subseteq X^\star$. Definimos*

$$\|V\| = \sum_{\sigma \in \text{pff}(V)} |\sigma|.$$

O resultado a seguir é útil nas computações.

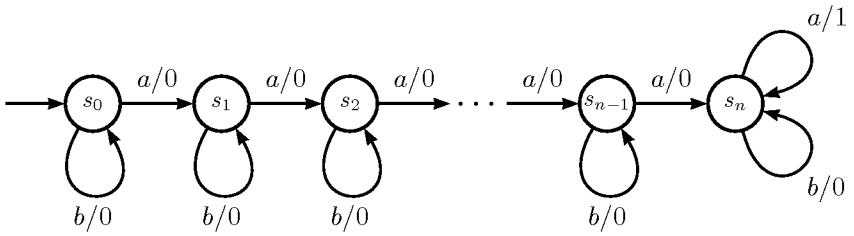


Figura 4: A especificação M_n.

Lemma 32 *Seja $V_1, V_2 \subseteq X^\star$, com V_2 finito. Então* $\text{pff}(V_1V_2) = \text{pff}(V_1 \cdot \text{pff}(V_2))$. •

Claramente, todos os conjuntos de teste práticos são finitos, assim satisfazendo as condições do lema e permitindo a computação de $\text{pff}(V_1V_2)$ pelo cálculo de $\text{pff}(V_1 \cdot \text{pff}(V_2))$ o qual é potencialmente menor. Contudo, o próximo exemplo mostra que a condição de finitude sobre V_2 pode não ser relaxada.

Example 33 *Seja $V_1 = \{ab, a\}$ e $V_2 = \{c\} \cup \{bca^i \mid i \geq 0\}$.*

Então $V_1V_2 = \{ac, abc\} \cup \{ab^2ca^i, abca^i \mid i \geq 0\}$, e temos $\text{pff}(V_1V_2) = \{ac\}$. *Também,* $\text{pff}(V_2) = \{c\}$ *e então* $V_1 \cdot \text{pff}(V_2) = \{abc, ac\} = \text{pff}(V_1 \cdot \text{pff}(V_2))$.

Sendo $\sigma = abc$ temos $\sigma \in \text{pff}(V_1 \cdot \text{pff}(V_2))$ e $\sigma \notin \text{pff}(V_1V_2)$. Isso mostra que $\text{pff}(V_1 \cdot \text{pff}(V_2)) \not\subseteq \text{pff}(V_1V_2)$ *quando V_2 não é finito.* •

Agora, considere a especificação ilustrada na Figura 4, onde $n \geq 0$ e $X = \{a, b\}$. Claramente, a máquina tem $n + 1$ estados. Seja $\rho = a^{n+1}$. Temos $\widehat{\lambda}(\rho, s_i) = 0^{n-i}1^{i+1}$, para todo i, $0 \leq i \leq n$. Daqui, $\widehat{\lambda}(\rho, s_i) \neq \widehat{\lambda}(\rho, s_j)$ se $i \neq j$, e podemos concluir que $s_i \not\approx s_j$ neste modelo, para todo i, j, $i \neq j$, e $0 \leq i, j \leq n$. Isso mostra que M_n é uma FSM mínima com índice $n + 1$.

Podemos pegar $W = \{a^{n+1}\}$ como um conjunto caracterização de M_n. Note que, além disso, $\|W\|$ é mínimo entre todos os conjuntos caracterização de M_n. Uma simples aplicação da construção da árvore rotulada proposta em [8] dá uma cobertura de transição para M_n:

$$P = \{\epsilon\} \cup \{a^ia, a^ib \mid 0 \leq i \leq n\}.$$

O modelo de implementação, M'_n, é obtido mudando a saída de apenas uma transição no último estado de M_n. Veja a Figura 5. Pegamos $m + 1 = \lceil \alpha \cdot (n + 1) \rceil$, para algum parâmetro $\alpha > 1$. Isto é, o modelo de implementação tem até $100(\alpha - 1)\%$ mais estados do que a especificação M_n. Então, se quisermos testar implementações com até 5% de estados a mais, tomamos $\alpha = 1.05$. Novamente, vemos facilmente que M'_n é uma FSM mínima com índice

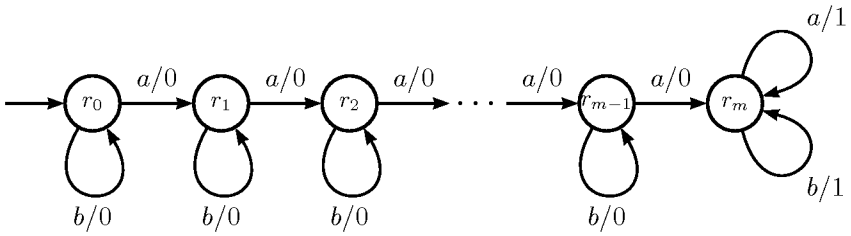


Figura 5: A implementação M'_n.

$m+1$.

Considere o conjunto de sequências de entrada $R = \{a^m b\}$. Então $\widehat{\lambda'}(a^m b, r_j) = 0^{m-j}1^{j+1}$, para todo j, $0 \leq j \leq m$. Daqui, $r_i \not\approx_R r_j$ no modelo de implementação M'_n para todo i, j com $i \neq j$ e $0 \leq i, j \leq m$. Assim, R particiona os estados de M'_n em $k = m+1$ classes de equivalência.

Agora, podemos comparar os conjuntos de teste que são obtidos usando o método G e o método W. Indicamos os conjuntos Z construídos pelos métodos G e W por Z_R e Z_W, respectivamente. Mostramos que a taxa $\frac{\|PZ_W\|}{\|PZ_R\|}$ cresce exponencialmente mais rápido, quando n aumenta[1].

Temos

$$Z_R = \bigcup_{i=0}^{(m+1)-k} X^i R = \{\epsilon\} \cdot R = \{a^m b\}$$

$$PZ_R = \big(\{\epsilon\} \cup \{a^i a, a^i b \,|\, 0 \leq i \leq n\}\big) \cdot \{a^m b\} = \{a^m b\} \cup \{a^i a a^m b, a^i b a^m b \,|\, 0 \leq i \leq n\}.$$

Por isso, mesmo contando todas as sequências em PZ_R, temos

$$\|PZ_R\| \leq (m+1) + 2(m+2)\sum_{i=0}^{m} i = (m+1)^3 \leq (\alpha(n+1)+1)^3 \leq 8\alpha^3 n^3,$$

assumindo $n \geq 2$ e lembrando que $\alpha > 1$.

Sobre o modelo de especificação M_n computamos

$$Z_W = \bigcup_{i=0}^{(m+1)-(n+1)} X^i W = \bigcup_{i=0}^{\ell} X^i \{a^{n+1}\},$$

onde $\ell = \lceil (n+1)(\alpha-1) \rceil$. Então, todas as sequências na forma $a^n b\, \rho\, a^{n+1}$ estão em pff(PZ_W), para todo $\rho \in X^\ell$. Note que $a^n b$ está em P e tem tamanho máximo dentre todas as sequências.

[1] Usamos a notação padrão O e Omega grande da teoria de complexidade [37].

Assim, $a^n b\, \rho\, a^{n+1}$ tem tamanho máximo dentre todas as sequências em PZ_W. Daqui, podemos escrever que $\|PZ_W\| \geq (2n + 2 + \ell)2^\ell \geq n2^{n(\alpha-1)}$.

Agora, considere a taxa $Q_\alpha(n) = \frac{\|PZ_W\|}{\|PZ_R\|}$. Claramente, temos que $Q_\alpha(n)$ é $\Omega(2^{cn})$ para todo c tal que $n(\alpha - 1) - nc > 0$, isto é, para todo c tal que $c < \alpha - 1$.

Concluímos que a família de pares de especificações e implementações M_n e M'_n, respectivamente, é tal que o método G gera conjuntos de teste que são exponencialmente mais efetivos do que aqueles gerados usando o método W, desde que o parâmetro fixo α satisfaça $\alpha > 1$.

Note que esta condição pode, potencialmente, aparecer quando os modelos M_n e M'_n ocorrerem como submodelos em outras especificações e implementações, respectivamente. Note também que podemos mudar livremente o alvo para todo b na implementação M'_n e o resultado ainda valeria. Isto é, o método G gera conjuntos de teste exponencialmente mais curtos e podem ser usados para testar uma classe inteira de implementações. Similarmente, podemos mudar livremente o alvo para todas as transições b no modelo de especificação M_n e ainda obter o mesmo resultado.

Isto sugere uma alternativa na estratégia de teste. Usando conjuntos de teste menores produzidos pelo método G, com uma escolha sensata dos parâmetros, podemos testar um número grande de modelos do conjunto de implementações candidatas, assumindo um limite superior fixo para o número de estados nos modelos das implementações. No caso de algum modelo de implementação passar no primeiro conjunto de teste, um conjunto de teste mais justo poderia ser produzido para testar os modelos de implementações remanescentes num segundo passo. Para este segundo passo conjuntos de teste completos poderiam ser construídos, com um esforço extra, através do método G com parâmetros mais justos, usando o método W original, ou qualquer outro método com cobertura completa.

Mais detalhes e provas formais desse resultado podem ser encontrados no relatório técnico [8], intitulado “Exponentially more Succinct Test Suites”.

O próximo capítulo apresenta uma técnica de geração de casos de teste para o modelo TIOA, usando um novo método de discretização, além da noção de proposta de teste e produto síncrono.

Capítulo 3

Discretização dos TIOA para a Verificação de Comportamentos

Prólogo

A verificação de corretude em sistemas complexos através de métodos eficientes de geração de casos de teste tem sido amplamente pesquisada. Neste trabalho, usamos a noção de teste baseado em modelos, representado aqui pelo modelo TIOA, um autômato temporizado com entradas e saídas autônomas. Daí a importância de se construir ferramentas que lidem com modelos que capturem o comportamento de sistemas críticos de tempo real. Porém, o grande problema das técnicas e métodos de geração de casos de teste baseados em modelos formais, aplicado a sistemas complexos, é a explosão combinatória no número de estados, principalmente em sistemas com características temporais.

Com o objetivo de contornar tais problemas, este trabalho propõe uma forma de discretização alternativa, usando o conceito de autômatos *grid* [15, 16]. No entanto, nossa proposta desvia da discretização clássica usando regiões de relógio, onde uma relação rígida entre o número de relógios e a granularidade escolhida para o grid deve ser mantida. Mais detalhes sobre uso das clássicas regiões de relógio na discretização de modelos temporizados podem ser encontrados em [1, 2, 28], aplicados aos clássicos modelos de *timed automata*.

Nessa proposta permitimos uma flexibilidade maior na escolha da granularidade usada na discretização, proporcionando uma diminuição significativa no número de estados do grid produzido, e evitando a explosão do espaço de estado. Através desse novo método de discretização mostramos que um modelo TIOA pode então simular seu respectivo grid, e vice-versa, para garantir a conformidade dos resultados.

Na sequência apresentamos também uma estratégia para a geração e aplicação de casos de teste em implementações candidatas usando o modelo discretizado, construído com o novo método. Utilizamos a noção de proposta de teste para modelar propriedades específicas do

sistema a ser testado. O conceito de produto de TIOA também é usado para modelar o comportamento conjunto entre uma especificação e uma proposta de teste. Dessa forma, um autômato grid pode ser obtido do produto, permitindo então que um conjunto de casos de teste seja extraído do modelo discretizado. As sequências de teste podem assim ser aplicadas em implementações candidatas. Neste caso usamos uma estratégia de realimentação dos autômatos grid que permite a verificação das propriedades encontradas nas implementações do modelo original.

Este capítulo é composto pelo artigo, "A New Discretization Method for Verifying Timed System Compatible Behaviors", que propõe um novo método de discretização para o modelo TIOA. O objetivo é utilizar modelos discretizados para a extração de casos de teste de forma eficiente. O trabalho mostra como conseguir a discretização sem usar a noção clássica de regiões de relógio, diminuindo o número de estados do grid a ser manipulado. Além disso, apresentamos uma estratégia para o teste de conformidade com implementações candidatas através de sequências de teste extraídas do modelo discretizado. O artigo foi submetido para a revista *Software Testing, Verification and Reliability* (STVR) da Wiley InterScience.

SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Relia. 2007;00:1-7 (DOI: 10.1002/000)

Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/000

A New Discretization Method for Verifying Timed System Compatible Behaviors

Adilson Luiz Bonifácio[*,†] **Arnaldo Vieira Moura**[*,‡]

Computing Institute, University of Campinas, Campinas, P.O. 6176, Brazil

SUMMARY

Devising formal techniques and methods that can automatically generate test suites for timed systems has remained a challenge. In this work we use Timed Input/Output Automata (TIOA) as a formal specification model for timed systems. We propose and prove the correctness of a new and more general discretization method that can be used to obtain grid automata corresponding to specification TIOA, using almost any granularity of interest. Such flexibility to find a suitable granularity opens the possibility for a more compact construction of grid automata. We also show how test purposes can be used together with the specification TIOA in order to generate grid automata that captures the behavior of both the specification and the test purpose. From such grid automata one can algorithmically extract test suites that can be used to verify whether given implementations conform to the specification and reflect the properties modeled using test purposes.

Received 06 November 2009; Revised 06 November 2009

KEY WORDS: Timed systems; Discretization; Test purpose; Timed test case generation.

1 Introduction

In order to verify the correctness of computational systems, automatic test case generation methods have been intensively investigated. One of the most important and promising among such

[*]Correspondence to: Computing Institute, University of Campinas, Campinas, P.O. 6176, Brazil
[†]E-mail: adilson@ic.unicamp.br
[‡]E-mail: arnaldo@ic.unicamp.br
Contract/grant sponsor: CNPq; contract/grant number: 141978/2008-2

techniques is model-based testing [1, 2, 3, 4]. Although mathematical models of system requirements and formally specified system functionalities allow for some automation in the process of efficient generation of test suites, devising techniques and methods that properly deal with critical and real-time systems has remained a challenge.

In this work we deal with timed system specifications and, in particular, with Timed Input/Output Automata (TIOA) [5, 6] which are a variant of the classical timed automata model [7, 8, 9, 10]. TIOA models can be used to specify timed systems and their executions, allowing continuous time evolution and discrete guarded transitions to be represented in the models. In order to obtain manageable representations for such models we use the notion of grid automata [11, 9], and show how they can be automatically extracted from the original timed models.

Discretization has already been proposed in connection with timed automata, in the form of clock regions [7, 12]. Here, however, we propose a different and more general notion of discretization, using adjusted values and limiting boundaries. This new proposal allows for more possibilities when choosing the granularities of interest. In fact, depending on the accuracy of the physical system being modeled, a corresponding grid automaton can be obtained for almost any desired granularity. Therefore, we can choose more adequate granularities for the properties under test, while exercising more control over the state space that results from the grid construction. In contrast, other methods use a granularity that is a fixed function of the number of clock regions.

In the next sections we expose the relationship between a specification TIOA and the corresponding grid automaton that results when the new discretization method is applied to the former. We also present proofs of correctness showing that the TIOA homomorphically simulates the grid automaton, and vice-versa. This forms the basis that will allow for the automation of test case generation methods that use the grid automata as a basis.

In order to test implementations against given specifications, we use the notion of a test purpose [13, 14]. A test purpose is a particular kind of TIOA that can be used to model specific properties of the system under test. Given a specification TIOA and a test purpose, their joint behavior is captured by computing the synchronous product of both models. Once the product is obtained, we can apply the discretization method to it, thus obtaining a grid automaton. Test sequences can then be automatically extracted from the resulting grid automaton. Having the test suite, one can use conformance testing methods to verify whether implementations conform to the desired behaviors modelled by the original TIOA specification and the test purpose [1]. In particular, one can use a trace inclusion strategy [15] to verify compatible behaviors.

This work is organized as follows. In Section 2, we define the TIOA model and some other important concepts. In Section 3, we present the new discretization method. We start with some basic concepts concerning bounding values and limiting functions in Subsection 3.1. Then, the grid construction and its properties are presented in Subsection 3.2, establishing the correct-

ness of the homomorphic relationship between the TIOA and its corresponding grid automaton. Section 4 briefly discusses the process of generating timed test suites using the notions of test purpose and synchronous product. In Section 5 we discuss some related works. Finally, some concluding remarks appear in Section 6.

Mode details and omitted proofs of simple claims can be found in [16].

2 The TIOA model

In this section we define the Timed I/O Automata (TIOA) model. But first, we need the notions of timed words, clock variables and clock conditions.

2.1 Timed words

Time instants and time delays will be taken from the set of non-negative rationals,[1] $\mathbb{Q}_{\geq}$.

Definition 1 *Let Σ be an alphabet. A* timed word *over Σ is a finite sequence $\langle \sigma_1, \sigma_2, \cdots, \sigma_n \rangle$, where $n \geq 0$ and $\sigma_i \in \Sigma \cup \mathbb{Q}_{\geq}$, for all i, $1 \leq i \leq n$.* •

For example, let $\Sigma = \{a, b, c, d\}$. Then $\langle 1, b, 2.3, d, d, 5.1, a \rangle$ is a timed word over Σ, and so is $\langle c, 5, a, 4 \rangle$. The intended interpretation is that a timed word like $\langle 1, c, 2, a \rangle$ can be understood as a symbol c arriving after one time unit and a symbol a arriving at $1 + 2 = 3$ time units, counting from the start. A time word like $\langle 1, c, a \rangle$ can be interpreted as $\langle 1, c, 0, a \rangle$, saying that symbols c and a arrive at the same time, but with c preceding a. Also, syntactically, a timed word like $\langle 1, c, 2, 5, a \rangle$ is not the same as the timed word $\langle 1, c, 7, a \rangle$, although they convey the same intended information.

When there is no risk for confusion, we may also write $\sigma_1, \sigma_2, \ldots, \sigma_n$, or even $\sigma_1 \sigma_2 \ldots \sigma_n$, in place of $\langle \sigma_1, \sigma_2, \cdots, \sigma_n \rangle$. The empty timed word will be denoted by ε. The set of all timed words over Σ will be denoted by Ψ_Σ, or simply Ψ if there is no room for confusion. The concatenation of timed words follows the standard definition for the concatenation of finite sequences. That is, if $\psi_1 = \langle \sigma_1, \ldots, \sigma_n \rangle \in \Psi_\Sigma$ and $\psi_2 = \langle \sigma_{n+1}, \ldots, \sigma_{n+m} \rangle \in \Psi_\Sigma$, with $n, m \geq 0$, then $\psi_1 \cdot \psi_2 = \langle \sigma_1 \ldots \sigma_n, \sigma_{n+1}, \ldots, \sigma_{n+m} \rangle$. We may also write $\psi_1 \psi_2$ instead of $\psi_1 \cdot \psi_2$. As an example, let $\psi_1 = \langle 1, a, 2, 5, b, c, 3 \rangle$ and $\psi_2 = \langle 4, a, d, 2 \rangle$. Then, we have $\psi_1 \cdot \psi_2 = \langle 1, a, 2, 5, b, c, 3, 4, a, d, 2 \rangle$.

[1] $\mathbb{Q}$ is the set of rationals, $\mathbb{Q}_{\geq}$ is the set of non-negative rationals and $\mathbb{Q}_{>}$ is the set of positive rationals.

2.2 Clocks

We will also need a set of clock variables, or clocks for short, denoted by C. The set of all clock conditions, Φ_C, is comprised by all expressions δ that can be finitely generated using the rules

$$\delta := \texttt{true} \mid c \leq \tau \mid \tau \leq c \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where c is a clock variable and $\tau \in \mathbb{Q}_{\geq}$ is a time instant. We will take the usual liberties when writing clock conditions, *e.g.*, we may write $c \geq \tau$ for $\tau \leq c$, or $c < \tau$ instead of $\neg(\tau \leq c)$, or $\tau_1 \leq c \leq \tau_2$ for $(\tau_1 \leq c) \wedge (c \leq \tau_2)$. A clock interpretation over C is a partial function from C into $\mathbb{Q}_{\geq}$. A total clock interpretation over C is a clock interpretation over C whose domain[2] is C. The set of all clock interpretations over C will be denoted by $[C \rightharpoonup \mathbb{Q}_{\geq}]$, and $[C \to \mathbb{Q}_{\geq}]$ will denote the set of all total clock interpretations over C. Clearly, $[C \rightharpoonup \mathbb{Q}_{\geq}] \subseteq [C \to \mathbb{Q}_{\geq}]$. When the intended set C is clear from the context, we may write clock interpretation, or simply interpretation, instead of clock interpretation over C.

Let $\delta \in \Phi_C$ and let $\nu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be such that all clock variables occurring in δ are in $\text{dom}(\nu)$. Then we say that ν satisfies δ, denoted by $\nu \models \delta$, if δ evaluates to true when every clock c is replaced by $\nu(c)$ in δ and the value of the resulting propositional logic sentence is computed in the usual manner.

Let $\nu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be an interpretation and let $\tau \in \mathbb{Q}_{\geq}$ be a time delay. The interpretation $\nu + \tau$ is defined as

$$(\nu + \tau)(c) = \begin{cases} \nu(c) + \tau & \text{if } c \in \text{dom}(\nu) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let $\mu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$. We define the interpretation $\nu \oplus \mu$ thus

$$(\nu \oplus \mu)(c) = \begin{cases} \mu(c) & \text{if } c \in \text{dom}(\mu) \\ \nu(c) & \text{if } c \in (\text{dom}(\nu) - \text{dom}(\mu)) \\ \text{undefined} & \text{if } c \notin (\text{dom}(\nu) \cup \text{dom}(\mu)), \end{cases}$$

for all $c \in C$. That is, $\nu \oplus \mu$ assigns clock values first according to μ and, baring that, then it assigns values according to ν, if at all possible. Note that when ν or μ is total, then so is $\nu \oplus \mu$.

2.3 Timed Input Output Automata

The Timed I/O Automata (TIOA) model is based on the BTDA (*Bounded Time Domain Automaton*) model of Gawlick et al [5]. A BTDA is composed by states, action symbols, clock variables, state invariants and transitions. The system progresses by a sequence of continuous

[2] $\text{dom}(f)$ will denote the domain of a function f.

time evolutions interrupted by discrete transitions [8, 17, 18]. During a continuous time evolution, the state does not change and its invariant must stay verified at all instants. A discrete transition is specified by a source and a target state, an action symbol, a transition guard and a (partial) clock reset function. We now make these concepts precise.

Definition 2 *A BTDA is a tuple* $(S, s_0, \Sigma, C, \nu_0, Inv, T)$*, where* S *is a finite set of states,* $s_0 \in S$ *is the initial state,* Σ *is the set of action symbols,* C *is a set of clocks,* $\nu_0 \in [C \to \mathbb{Q}_{\geq}]$ *is the initial clock interpretation, where* $\nu_0(c) = 0$ *for all* $c \in C$*,* $Inv : S \to \Phi_C$ *maps states to state invariants, and* T *is the set of transitions, where* $T \subseteq (S \times \Sigma \times \Phi_C \times [C \curvearrowright \mathbb{Q}_{\geq}] \times S)$*.* •

The intended meaning for a transition $(s, z, \delta, \theta, r)$ is that the machine can move to state r from state s over the symbol z provided that the guard δ is enabled. Further, upon moving to state r, the mapping $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$ indicates which clocks are reset and to which values. Note that the formalism allows for clocks to be reset to any values, not just to zero.

As an example, Figure 1 shows a BTDA with one clock variable, c, and two action symbols, $\Sigma = \{on, off\}$. The set of states is $S = \{q_0, q_1\}$, with the initial state q_0 being marked by an incoming arrow with no source node. The invariants are indicated right next to the corresponding state; for example, $Inv(q_0)$ is $c \leq K$, where we take $K > 5$. The remaining arrows indicate transitions. Next to each arrow we can see the corresponding action symbol, the logical expression is the transition guard and the attribution like notation indicates the clock resetting partial map. For example, the transition from q_0 to q_1 would be written as $(q_0, on, c \leq K, \theta, q_1)$, where $\theta(c) = 0$. In this transition the only clock is reset to zero. However we emphasize that

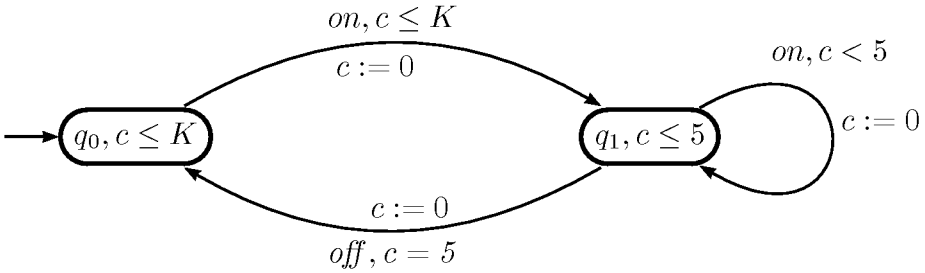


Figure 1: A BTDA model of a simple switch.

the clock variables can be reset to different values. This generalization is benefit when the aim is to assign different values to different clock variables over a transition.

From now on, B will always denote a BTDA $B = (S, s_0, \Sigma, C, Inv, T)$, and we let any decorations carry over uniformly to the components of B, *e.g.*, $B' = (S', s_0', \Sigma', C', Inv', T')$.

In order to specify the movements of a BTDA, we need the notion of a configuration. A configuration of a BTDA B is a pair (s, ν), where $s \in S$ is a state and $\nu \in [C \to \mathbb{Q}_{\geq}]$ is a total clock interpretation over C. The set of all configurations of B is denoted by Γ_B, or simply by Γ if no confusion can arise. The initial configuration is (s_0, ν_0), where $\nu_0(c) = 0$ for all $c \in C$. We require that $\nu_0 \vDash Inv(s_0)$. In order to ease the notation, when the set of clocks is small we indicate a configuration simply by listing the corresponding state followed by a list of the clock values, in some pre-arranged order. For example, from Figure 1 we can see that (q_0, ν) is a configuration, where $\nu(c) = 1.5$. We may also write $(q_0, 1.5)$ to indicate this same configuration.

We are now ready to specify the elementary moves of a BTDA.

Definition 3 *Let B be a BTDA and let $\gamma_i = (s_i, \nu_i) \in \Gamma_B$, $i = 1, 2$, be two configurations of B. We define:*

1. *Let $\tau \in \mathbb{Q}_{\geq}$ be a time delay. Then there exists a* continuous movement *from γ_1 to γ_2 over τ, denoted by $\gamma_1 \underset{\tau}{\to} \gamma_2$, if and only if: (i) $s_1 = s_2$; (ii) $\nu_2 = \nu_1 + \tau$; and (iii) $\nu_1 + \eta \vDash Inv(s_1)$ for all η, $0 < \eta \leq \tau$. When τ is positive, we have a* non-trivial continuous movement.

2. *Let $x \in \Sigma$ be an action symbol. Then there exists a* discrete movement *from γ_1 to γ_2 over x, denoted by $\gamma_1 \overset{x}{\to} \gamma_2$, if and only if there exists a transition $(s_1, x, \delta, \theta, s_2) \in T$ such that: (i) $\nu_1 \vDash \delta$; (ii) $\nu_2 = \nu_1 \oplus \theta$; and (iii) $\nu_2 \vDash Inv(s_2)$.* •

In a discrete movement the state changes while the clock interpretation remains fixed, except for a subset of the clocks that are reset as indicated by the partial function θ. Moreover, we also require that the corresponding transition guard is enabled and that the new clock interpretation satisfies the invariant of the target state. By way of contrast, in a continuous movement, the state does not change and the clock interpretation absorbs the delay τ. Note that the state invariant must remain satisfied during the whole period of the continuous movement. As an example, from Figure 1 we have: $(q_0, 0) \overset{on}{\to} (q_1, 0)$, $(q_1, 0) \underset{5}{\to} (q_1, 5)$, $(q_1, 5) \overset{off}{\to} (q_0, 0)$, and $(q_0, 0) \underset{1}{\to} (q_0, 1)$.

Starting from a configuration, a timed word induces a movement in a BTDA.

Definition 4 *Let B be a BTDA. The* movement relation *of B, $\underset{B}{\vdash}$, is a binary relation over $\Psi_\Sigma \times \Gamma_B$ given by $(\psi_1, \gamma_1) \underset{B}{\vdash} (\psi_2, \gamma_2)$ if and only if $\psi_1 = \langle \sigma \rangle \cdot \psi_2$ and either (i) $\sigma \in \mathbb{Q}_{\geq}$ and $\gamma_1 \underset{\sigma}{\to} \gamma_2$; or (ii) $\sigma \in \Sigma$ and $\gamma_1 \overset{\sigma}{\to} \gamma_2$.* •

The k-th power of $\underset{B}{\vdash}$ will be indicated by $\overset{k}{\underset{B}{\vdash}}$, $k \geq 0$, and its reflexive transitive closure by $\overset{\star}{\underset{B}{\vdash}}$. As usual, we may drop decorations when no confusion can arise.

As an example, consider the timed word $\psi = \langle K/2, on, 3, on, 5, \mathit{off}, K/3\rangle$ and the BTDA in Figure 1. Then, we have

$$\begin{aligned}\big(\psi, (q_0, 0)\big) &\vdash \big(\langle on, 3, on, 5, \mathit{off}, K/3\rangle, (q_0, K/2)\big) \vdash \big(\langle 3, on, 5, \mathit{off}, K/3\rangle, (q_1, 0)\big)\\ &\vdash \big(\langle on, 5, \mathit{off}, K/3\rangle, (q_1, 3)\big) \vdash \big(\langle 5, \mathit{off}, K/3\rangle, (q_1, 0)\big)\\ &\vdash \big(\langle \mathit{off}, K/3\rangle, (q_1, 5)\big) \vdash \big(\langle K/3\rangle, (q_0, 0)\big) \vdash \big(\varepsilon, (q_0, K/3)\big).\end{aligned}$$

Further, when $(\psi, \gamma) \overset{\star}{\vdash} (\varepsilon, \rho)$ we also write $\gamma \overset{\psi}{\Vdash} \rho$, or $\gamma \Vdash \rho$ when the particular timed word is not relevant.

When $\gamma \overset{\psi}{\Vdash} \rho$ we say that ψ is a *run starting* at γ and *ending* at ρ. If, moreover, γ is the initial configuration of the BTDA, then ψ is an *execution ending* at ρ. A configuration γ is *reachable* if there is an execution ending at γ. Likewise, a state s is *reachable* if there is a reachable configuration (s, ν), for some interpretation ν. Clearly, as can be seen from Figure 1, we can write $(q_0, 0) \overset{\langle on, 5\rangle}{\Vdash} (q_1, 5)$, thus configuration $(q_1, 5)$ and state q_1 are reachable. Also, it is easy to see that configuration $(q_1, 6)$ is not reachable.

Timed I/O Automata [6] extend the BTDA model by partitioning the set of actions into input and output actions. Input actions are considered stimulus from the environment while output actions will be generated by the system.

Definition 5 *A* Timed Input/Output Automaton *is a triple $M = (B, X, Y)$ where:*

1. *$B = (S, s_0, \Sigma, C, \nu_0, Inv, T)$ is a BTDA, the subjacent BTDA of M, and*
2. *$\{X, Y\}$ partitions Σ into a set X of input actions and a set Y of output actions.* •

By letting $X = \{on\}$ and $Y = \{\mathit{off}\}$ in Figure 1 we partition the actions of the BTDA, thereby obtaining a TIOA. From now on, M will always denote the TIOA (B, X, Y) and any decorations will carry over uniformly to the components of M. The set of configurations of M, Γ_M, as well as the notion of movements of M are taken directly from the corresponding notions for the subjacent BTDA B, as in Definitions 2 and 3.

3 TIOA Discretization

The number of test sequences for any TIOA is infinite, since these sequences are extracted from distinct runs. Clearly, it is not practical to consider the set of all such sequences when putting a system to the test. An alternative is to use some sort of discretization of the original TIOA. Often, such discretizations lead to the notion of a grid automaton [11, 9] that is obtained using the notion of clock regions and sampling graphs. We propose an alternative method for

discretizing TIOA, one that does not use the classical notions of clock regions and sampling graphs. We note that with this new approach one can choose any discrete granularity in the form $1/k$, where k is a positive integer, independently of the number of clock regions in the original TIOA.

We start by considering bounded values and bounded functions.

3.1 Bounded values and functions

In what follows, given $t \in \mathbb{Q}_{\geq}$, we will denote the integral and fractional parts of t by $\lfloor t \rfloor$ and $\lceil t \rceil$, respectively. Hence, $t = \lfloor t \rfloor + \lceil t \rceil$ always holds. We note the following simple facts.

Fact 6 *Let $x, y, z \in \mathbb{Q}_{\geq}$ and let k a positive integer. The following hold:*

1. *If $x \geq y$ then $\lfloor x \rfloor \geq \lfloor y \rfloor$.*
2. *If $x = y + k$ then $\lceil x \rceil = \lceil y \rceil$ and $\lfloor x \rfloor = \lfloor y \rfloor + k$.*
3. *If $x = y + z$ then $\lceil x \rceil = \left\lceil \lceil y \rceil + \lceil z \rceil \right\rceil$ and $\lfloor x \rfloor = \lfloor y \rfloor + \lfloor z \rfloor + \left\lfloor \lceil y \rceil + \lceil z \rceil \right\rfloor$.*
4. *If $\lceil x \rceil = \lceil y \rceil$ then $\lceil x + z \rceil = \lceil y + z \rceil$.*

Proof Immediate from the definitions. •

Next, we turn to the notion of bounded interpretations.

Definition 7 *Let $L \in \mathbb{Q}_{\geq}$ be a bound. Let $x \in \mathbb{Q}_{\geq}$ be a value, let A be a set and let $\alpha \in [A \rightarrow \mathbb{Q}_{\geq}]$ be a function. We define*

1. *The L-bounded x value, denoted x_L, is given by*

$$x_L = \begin{cases} x & \text{if } x \leq L \\ \lfloor L \rfloor + \lceil x \rceil & \text{otherwise. } \bullet \end{cases}$$

2. *The L-bounded α function, denoted α_L, is obtained by letting $\alpha_L(a) = (\alpha(a))_L$, for all $a \in A$.*

The following simple facts are immediate.

Fact 8 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x \in \mathbb{Q}_{\geq}$ be a value. Then:*

1. *$\lceil x_L \rceil = \lceil x \rceil$ and $x_L \leq x$.*
2. *If $x < \lfloor L \rfloor + 1$ then $x_L = x$.*

3. $x_L < \lfloor L \rfloor + 1$.

4. $(x_L)_L = x_L$.

Proof Straightforward from the definitions and from Fact 6. •

The next proposition essentially states that the L-bound of a sum of terms is the same as the L-bound of the sum of the L-bounds of the individual terms.

Proposition 9 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x, y \in \mathbb{Q}_{\geq}$. Then*

1. $(x + y)_L = (x + y_L)_L$.

2. $(x + y)_L = (x_L + y_L)_L$.

3. *If $x' = x$ or $x' = x_L$ and $y' = y$ or $y' = y_L$, then $(x + y)_L = (x' + y')_L$.*

Proof Follow easily from the definitions and from Fact 8. •

We can now extend these results to larger sums.

Proposition 10 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $x^i \in \mathbb{Q}_{\geq}$ be values, for $i = 1, \ldots, n$, with $n \geq 1$. Let $y^i = x^i$ or $y^i = (x^i)_L$, for $i = 1, \ldots, n$. Then $\left(\sum_{i=1}^n x^i\right)_L = \left(\sum_{i=1}^n y^i\right)_L$.*

Proof An easy induction on n, using Fact 8 and Proposition 9. •

A similar result holds when considering bounding functions.

Proposition 11 *Let $L \in \mathbb{Q}_{\geq}$ be a bound. Let $W = \{w_1, \ldots, w_n\}$ be a set, with $n \geq 1$. Also let $k_i \geq 0$ be integer constants, $i = 1, \ldots, n$. Take $\alpha \in [W \to \mathbb{Q}_{\geq}]$. Then $\left[\sum_{i=1}^n k_i \alpha(w_i)\right]_L = \left[\sum_{i=1}^n k_i \alpha_L(w_i)\right]_L$.*

Proof Using Proposition 10 we get $\left[\sum_{i=1}^n k_i \alpha(w_i)\right]_L = \left[\sum_{i=1}^n \left(k_i \alpha(w_i)\right)_L\right]_L$. Now, $\left[k_i \alpha(w_i)\right]_L = \left[\sum_{j=1}^{k_i} \alpha(w_i)\right]_L$ and so, using Proposition 10 again, we obtain

$$\left[k_i \alpha(w_i)\right]_L = \left[\sum_{j=1}^{k_i} (\alpha(w_i))_L\right]_L = \left[k_i \alpha_L(w_i)\right]_L.$$

Putting it together, we have $\left[\sum_{i=1}^n k_i \alpha(w_i)\right]_L = \left[\sum_{i=1}^n \left(k_i \alpha_L(w_i)\right)_L\right]_L$. Finally, using Proposition 10 once more, we get $\left[\sum_{i=1}^n k_i \alpha(w_i)\right]_L = \left[\sum_{i=1}^n k_i \alpha_L(w_i)\right]_L$, as desired. •

In order to obtain a finite number of clock interpretations in the grid automata, to be defined shortly, we impose an upper bound on clock values. Consider a set of clocks C and a time instant $L \in \mathbb{Q}_{\geq}$. Then, for any clock interpretation $\nu \in [C \to \mathbb{Q}_{\geq}]$, the L-bounded clock interpretation $\nu_L \in [C \to \mathbb{Q}_{\geq}]$ is constructed as in Definition 7. This notion is inspired by the idea of clock regions [7, 19]. We also refer to ν_L as the clock interpretation ν bounded by L. Note that we could have specified a different bound L_c for each clock c. In order to keep the notation uncluttered, however, we will consider a single bound L for all clock variables. It should be a simple matter to generalize all results to the case when some clock bounds may be distinct.

The next proposition expresses the result of L-bounding time delays and clock resets.

Proposition 12 *Let C be a set of clocks, let $\nu \in [C \to \mathbb{Q}_{\geq}]$ and and let L be a positive integer. We have:*

1. *If $\eta \in \mathbb{Q}_{\geq}$, then $(\nu + \eta)_L = (\nu_L + \eta)_L$.*
2. *If $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$, then $(\nu \oplus \theta)_L = (\nu_L \oplus \theta)_L$.*

Proof Using Proposition 11, it is easy to show that $(\nu + \eta)_L(c) = (\nu_L + \eta)_L(c)$, for all $c \in C$. And, using Fact 8 it is easy to show that $(\nu \oplus \theta)_L(c) = (\nu_L \oplus \theta)_L(c)$, for all $c \in C$. •

In the next two lemmas we present some useful relationships between limited interpretations, interpretations and evaluating conditions.

Lemma 13 *Let C be a set of clocks and let α^i, $\beta^i \in [C \to \mathbb{Q}_{\geq}]$, $i = 1, 2$, be clock interpretations. Assume that $\beta^i = \alpha^i$ or $\beta^i = (\alpha^i)_L$ for all $i \in \{1, 2\}$. Further, let $I \in \Phi_C$ be a clock condition and let L be a positive integer greater than all constants occurring in I. Then $\alpha^1 + \alpha^2 \vDash I$ iff $\beta^1 + \beta^2 \vDash I$.*

Proof If I is `true` we are done. Assume now that I is not `true`.

From Fact 6 we know that $\beta^i(c) \leq \alpha^i(c)$, for all $c \in C$ and all $i \in \{1, 2\}$.

We treat first the four simple cases when I is $(c \leq \tau)$, $(c \geq \tau)$, $\neg(c \leq \tau)$ or $\neg(c \geq \tau)$.

CASE 1: I is $(c \leq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$.

If $\alpha^1 + \alpha^2 \vDash I$ then $(\alpha^1 + \alpha^2)(c) = \alpha^1(c) + \alpha^2(c) \leq \tau$. Then $\beta^1(c) + \beta^2(c) \leq \tau$ and so $\beta^1 + \beta^2 \vDash I$.

For the converse, assume $\beta^1(c) + \beta^2(c) \leq \tau$. From the hypothesis, $\beta^1(c) + \beta^2(c) \leq L$.

If $\alpha^1(c) > L$ then either (i) $\beta^1(c) = \alpha^1(c) > L$, or (ii) $\beta^1(c) = (\alpha^i(c))_L = \lfloor L \rfloor + \lceil \alpha^1(c) \rceil = L + \lceil \alpha^1(c) \rceil$, since L is an integer. In any case, we contradict $\beta^1(c) + \beta^2(c) \leq L$.

Similarly, we cannot have $\alpha^2(c) > L$. Hence, $\alpha^i(c) \leq L$ and we get $\beta^i(c) = \alpha^i(c)$, for $i = 1, 2$. Then, since $\beta^1(c) + \beta^2(c) \leq \tau$, we also get $\alpha^1(c) + \alpha^2(c) \leq \tau$, and so $\alpha^1 + \alpha^2 \vDash I$.

CASE 2: I is $(c \geq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$.

First, let $\beta^1 + \beta^2 \models I$. Then $\beta^1(c) + \beta^2(c) \geq \tau$ and so $\alpha^1(c) + \alpha^2(c) \geq \tau$, since $\alpha^i(c) \geq \beta^i(c)$, $i = 1, 2$. Then, $\alpha^1 + \alpha^2 \models I$.

For the converse, assume $\alpha^1(c) + \alpha^2(c) \geq \tau$. If $\alpha^1(c) \geq L + 1$, then $\beta^1(c) = \lfloor L \rfloor + \lceil \alpha^1(c) \rceil = L + \lceil \alpha^1(c) \rceil$, since L is an integer. Thus, $\beta^1(c) \geq \tau$, since $L > \tau$ from the hypothesis. Hence $\beta^1(c) + \beta^2(c) \geq \tau$ and so $\beta^1 + \beta^2 \models I$.

Similarly, when $\alpha^2(c) \geq L + 1$ the result also holds.

Now let $\alpha^i(c) < L + 1 = \lfloor L \rfloor + 1$ for $i = 1, 2$. From Fact 8, we get $\beta^i(c) = \alpha^i(c)$ and so $\beta^1(c) + \beta^2(c) \geq \tau$, and again $\beta^1 + \beta^2 \models I$.

CASE 3: I is $\neg(c \leq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$. Equivalently, we have $\alpha^1 + \alpha^2 \models (c > \tau)$. We proceed as in Case 2.

CASE 4: I is $\neg(c \geq \tau)$, for some $c \in C$ and some $\tau \in \mathbb{Q}_{\geq}$. Equivalently, we have $\alpha^1 + \alpha^2 \models (c < \tau)$. We proceed as in Case 1.

Let $n \geq 0$ be the number of propositional connectives occurring in I. We proceed by induction on n.

BASIS: $n = 0$. The result holds by Cases 1 and 2.

INDUCTION STEP: assume the result holds for all I with at most n propositional connectives, where $n \geq 0$. Now, take some $I \in \Phi_C$ with $n + 1$ propositional connectives. We have two cases:

Case I-1: I is $\delta_1 \wedge \delta_2$.

Since δ_i has at most n propositional connectives, $i = 1, 2$, from the induction hypothesis we get $\alpha^1 + \alpha^2 \models \delta_i$ iff $\beta^1 + \beta^2 \models \delta_i$, for $i = 1, 2$. Then, clearly, $\alpha^1 + \alpha^2 \models I$ iff $\beta^1 + \beta^2 \models I$.

Case I-2: I is $\neg\delta$.

Then δ has $n \geq 0$ propositional connectives. When $n = 0$, δ is $(c \leq \tau)$ or $(c \geq \tau)$, and the result follows by Cases 3 and 4, respectively.

Assume now that $n \geq 1$. We have two sub-cases:

I-2A: δ is $\neg\delta_1$ and δ_1 has $n - 1$ propositional connectives. Then $\neg\delta$ is equivalent to $\neg\neg\delta_1$, that is, I is equivalent to δ_1. We can use the induction hypothesis and conclude that $\alpha^1 + \alpha^2 \models I$ iff $\beta^1 + \beta^2 \models I$.

I-2B: δ is $(\delta_1 \wedge \delta_2)$, that is, I is equivalent to $(\neg\delta_1) \vee (\neg\delta_2)$. Note that δ has n propositional connectives. Then δ_i has at most $(n-1)$ propositional connectives, that is, $\neg\delta_i$ has at most n propositional connectives, for $i = 1, 2$. Using the induction hypothesis we get $\alpha^1 + \alpha^2 \vDash \neg\delta_i$ iff $\beta^1 + \beta^2 \vDash \neg\delta_i$. Thus, $\alpha^1 + \alpha^2 \vDash (\neg\delta_1) \vee (\neg\delta_2)$ iff $\alpha^1 + \alpha^2 \vDash \neg\delta_i$ for some $i \in \{1, 2\}$ iff $\beta^1 + \beta^2 \vDash \neg\delta_i$ for some $i \in \{1, 2\}$ iff $\beta^1 + \beta^2 \vDash (\neg\delta_1) \vee (\neg\delta_2)$, completing the proof. •

Lemma 14 *Let C be a set of clocks and let $\nu \in [C \to \mathbb{Q}_{\geq}]$ and $\theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be clock interpretations and let $I \in \Phi_C$ be a clock condition. If $\nu \oplus \theta \vDash I$ then $\nu_L \oplus \theta \vDash I$, when L is a positive integer greater than all constants occurring in I.*

Proof The argument is very similar to the proof of Lemma 13, but now treating clock resets instead of clock displacements. Full details can be found in [16]. •

3.2 Grid automata

Given a TIOA M, in order to deal with a finite set of useful interpretations we choose a time granularity and discretize the movements of M. The granularity has to be chosen in a such way that neither it is too small, rendering the system almost continuous, nor is it too large, making the approximations too coarse. Some works [11, 9, 12] consider a discretization step of $1/(n+1)$, if $n = 1$, and $1/(n+2)$, with $n \geq 2$, where n is the number of clocks in M. In this work we show how to use any granularity in the form $g = 1/k$, k being a positive integer, and still obtain an adequate discretization. That is, even when choosing coarser granularities, once a bound L is also chosen, the behavior of the original TIOA is still homomorphically reproducible in the resulting grid, and conversely. Clearly, by using a granularity $1/k$, with $k << (n+1)$, the resulting grid automaton will, potentially, have a much smaller number of grid states, thereby significantly reducing the state space explosion problem.

We start with the notion of adjusted values.

Definition 15 *Let $g \in \mathbb{Q}_{\geq}$. Then*

1. *A value $\ell \in \mathbb{Q}_{\geq}$ is* g-adjusted *iff ℓ is an integer multiple of g.*
2. *Let C be a set of clocks. A clock condition $\delta \in \Phi_C$ is* g-adjusted *iff all constants occurring in δ are g-adjusted values. A clock interpretation $\nu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ is* g-adjusted *iff $\nu(c)$ is a g-adjusted value, for all $c \in dom(\nu)$.* •

When the value g can be inferred from the context, we may write simply adjusted instead of g-adjusted.

Two following propositions say that extending g-adjusted clock interpretations, or resetting a g-adjusted clock interpretation, always results in a new clock interpretation that is also g-adjusted.

Proposition 16 *Let $g = 1/k$, with k a positive integer, let $L \in \mathbb{Q}_{\geq}$ be a g-adjusted value, and let C be a set of clocks. Then*

1. *A value $\ell \in \mathbb{Q}_{\geq}$ is g-adjusted iff both $\lfloor \ell \rfloor$ and $\lceil \ell \rceil$ are g-adjusted values.*
2. *If $\nu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ is a g-adjusted clock interpretation, then ν_L is also a g-adjusted clock interpretation.*
3. *If $\nu, \eta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ are g-adjusted clock interpretations and ℓ is a g-adjusted value, then $\nu + \ell$ and $\nu \oplus \eta$ are also g-adjusted clock interpretations.*

Proof Follows without effort from the definitions. •

Next, we show that any set of L-bounded clock interpretations is finite, provided that L is properly adjusted.

Lemma 17 *Let C be a set of clocks, and let $g = 1/k$ with k a positive integer. Let $R \subseteq [C \rightharpoonup \mathbb{Q}_{\geq}]$ be a set of g-adjusted clock interpretations, and let $L \in \mathbb{Q}_{\geq}$ be a g-adjusted value. Consider the L-bounded set $R_L = \{\nu_L \mid \nu \in R\}$. Then $|R_L| \leq (k\lfloor L \rfloor + k)^{|C|}$.*

Proof Consider some $\nu_L \in R_L$ and some $c \in C$. Let $t = \nu_L(c)$. By Proposition 16, t is always g-adjusted. Moreover, $t < \lfloor L \rfloor + 1$, by Fact 8. But $\lfloor L \rfloor = ng$, for some non-negative integer n, and so $\lfloor L \rfloor + 1 = ng + (1/g)g = (n+k)g$. Hence, t can have at most $n+k$ distinct g-adjusted values (counting from zero).

We conclude that any clock $c \in C$ can be mapped to at most $n+k$ distinct g-adjusted values, by ν_L bounded interpretations. Therefore, there are at most $(n+k)^{|C|}$ distinct ν_L interpretations. Since $n = \lfloor L \rfloor / g = k\lfloor L \rfloor$, the result follows. •

Had we used a distinct g-adjusted bound L_c for each clock c, the lemma would yield the bound $|R_L| \leq \prod_{c \in C} (k\lfloor L_c \rfloor + k)$.

Before defining the grid automaton, we introduce the notions of adjusted timed words and adjusted TIOA.

Definition 18 *Let Σ be an alphabet. A timed word $\langle \sigma_1, \ldots, \sigma_n \rangle \in \Psi_\Sigma$ is g-adjusted iff σ_i is a g-adjusted value whenever $\sigma_i \in \mathbb{Q}_{\geq}$, for all i, $1 \leq i \leq n$.* •

The set of all g-adjusted timed words over Σ will be denoted by $\Psi_{g,\Sigma}$. As before, we may drop subscripts if there is no reason for confusion.

A TIOA will be said adjusted if both its guards and invariants are also adjusted. The notion of an adjusted TIOA will be important in reducing the set of L-bounded interpretations to a finite size.

Definition 19 *Let M be a TIOA. Then M is* g-adjusted *iff for all transitions $(s, z, \delta, \theta, r) \in T$ we have that δ is a g-adjusted clock condition and θ is a g-adjusted clock interpretation. Moreover, for all states $s \in S$, we require that $Inv(s)$ be a g-adjusted clock condition.* •

A run over a g-adjusted timed word implies the g-reachability of the terminal configuration.

Definition 20 *Let M be a g-adjusted TIOA, $s \in S$ and $\nu \in [C \to \mathbb{Q}_{\geq}]$. We say that (s, ν) is g-reachable in M iff there is a g-adjusted timed word $\psi \in \Psi_{g,\Sigma}$, such that $(s_0, \nu_0) \Vdash_M^{\psi} (s, \nu)$.* •

The result obtained by the discretization of a TIOA M will be a labeled transition system, called a grid automaton [9, 11].

Definition 21 *Let $g = 1/k$ with k a positive integer. Let M be a g-adjusted TIOA and let $L \in \mathbb{Q}_{\geq}$ be a g-adjusted value. Then the* L, g-grid automaton*, or simply* L, g-grid*, associated with M is the labeled transition system constructed by Algorithm 2.* •

Note that the input alphabet of the grid $M_{L,g}$ is formed by the set of all action symbols of M, together with the new symbol g. As usual, we may drop the qualifications L and g, when no confusion can arise.

Consider the TIOA depicted in Figure 2. Note that we have two clocks. In some earlier works [11, 9, 12] a granularity of $\frac{1}{4}$, or even smaller, must be used when discretizing a TIOA with two clocks. Here, instead, we may choose coarser values. In fact, for this example we will choose a granularity of $g = \frac{1}{2}$ and a bound $L = 6$. Further, let $K = 3$ in Figure 2. Under these assumptions, Figure 3 shows part of the grid automaton obtained from Algorithm 2, given Figure 2 as the input TIOA.

The next result establishes that Algorithm 2 always terminates. Moreover, it also gives an upper bound on the number of states in the resulting labeled transition system.

Lemma 22 *Consider the procedure depicted as Algorithm 2. Then, it always halts with* $|S_G| \leq |S| \times \big(k\lfloor L \rfloor + k\big)^{|C|}$.

Input: A value $g = 1/k$, with k a positive integer; a g-adjusted TIOA $M = (S, s, \Sigma, C, \nu, Inv, T)$, and a g-adjusted boundary $L \in \mathbb{Q}_{\geq}$.
Output: The L, g-grid $M_{L,g} = (S_G, s_G, \Sigma_G, T_G)$ associated with M.
begin
  $T_G \leftarrow \emptyset$ // the set of transitions;
  $RS \leftarrow s_G = (s, \nu)$ // where $\nu(c) = 0$ for all $c \in C$;
  $HS \leftarrow \emptyset$ // the set of visited states;
  while $RS \neq \emptyset$ **do**
    get a state (s, ν) from RS // choose a state;
    move (s, ν) from RS to HS;
    foreach $(s, z, \delta, \theta, r) \in T$ **do**
      if $\nu \vDash \delta$ *and* $\nu \oplus \theta \vDash Inv(r)$ **then**
        let $\eta = (\nu \oplus \theta)_L$;
        add the transition $((s, \nu), z, (r, \eta))$ to T_G ;
        add the state (r, η) to RS, if $(r, \eta) \notin HS$;
      end
    end
    if $\nu + h \vDash Inv(s)$ *for all* $0 < h \leq g$ **then**
      let $\eta = (\nu + g)_L$;
      add the transition $((s, \nu), g, (s, \eta))$ to T_G;
      add the state (s, η) to RS, if $(s, \eta) \notin HS$;
    end
  end
  $S_G \leftarrow HS$;
  $\Sigma_G \leftarrow \Sigma \cup \{g\}$;
  return;
end

Algorithm 2: Grid algorithm.

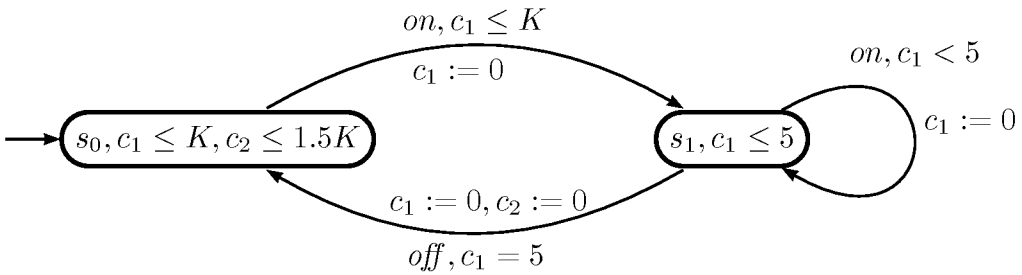


Figure 2: A TIOA model for a variation of the simple switch.

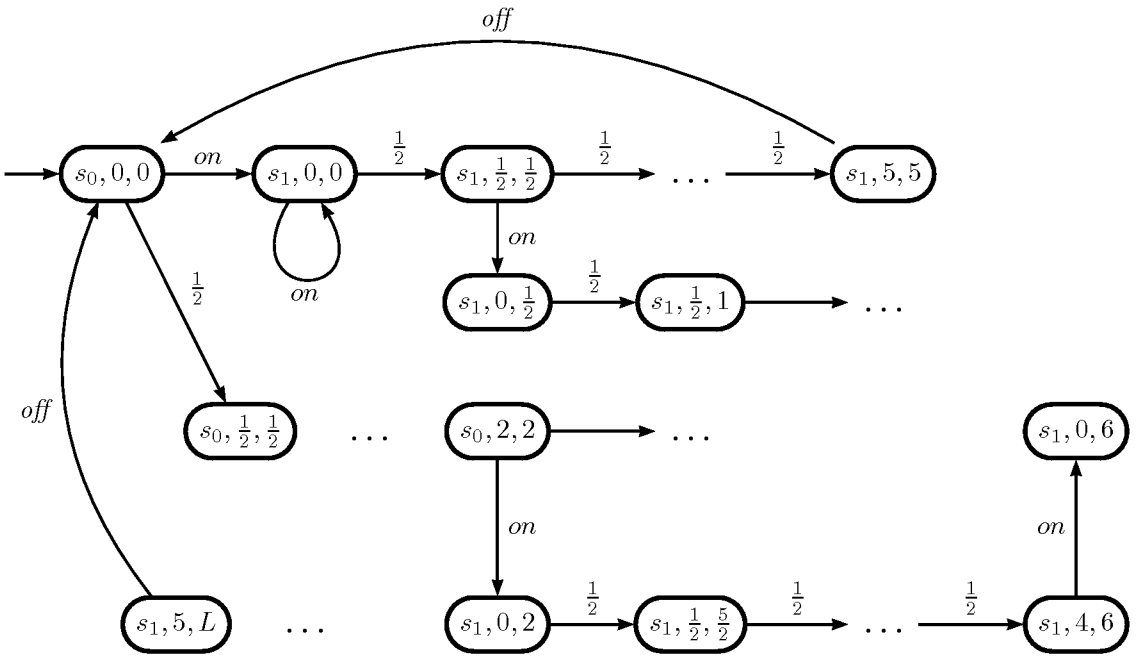


Figure 3: The partial grid automaton for Figure 2 (part 1 of 2).

Proof Since L is g-adjusted and $g = 1/k$, Lemma 17 states that there are at most $\big(k\lfloor L\rfloor + k\big)^{|C|}$ L-bounded distinct clock interpretations, that is, $|[C \curvearrowright \mathbb{Q}_{\geq}]_L| \leq \big(k\lfloor L\rfloor + k\big)^{|C|}$. Then, there are at most $|S| \times \big(k\lfloor L\rfloor + k\big)^{|C|}$ configurations of the form (s, ν_L), where $s \in S$ and ν_L is an L-bounded clock interpretation of M.

Notice that ν_0, at line 5, is also g-adjusted and L-bounded, trivially. Then, using Proposition 16 and Proposition 16 we know that the interpretations constructed at lines 12 and 18 are also g-adjusted. Clearly, then whenever a configuration (p, η) is added to RS at lines 14 and 20, we have that η is a g-adjusted clock interpretation.

Note that at lines 14 and 20 a state (s, η) is added to RS only if it is not already in HS. Thus, a state $(s, \eta) \in S \times [C \curvearrowright \mathbb{Q}_{\geq}]_L$ enters RS at most once. Since the loop at line 7 moves one state (s, ν) from RS into HS, we conclude that the loop at line 7 executes for at most $|S| \times (k\lfloor L\rfloor + k)^{|C|}$ times. Thus, clearly, the procedure halts and $|HS| \leq |S| \times |[C \curvearrowright \mathbb{Q}_{\geq}]_L|$. From line 23, we get $|S_G| \leq |S| \times \big(k\lfloor L\rfloor + k + 1\big)^{|C|}$. •

Note that, when we specify a possibly different bound L_c for each clock $c \in C$, the number of states in the grid automaton will be bounded by $k^{|C|} \times \prod_{c \in C} (\lfloor L_c\rfloor + k)$, which can be much

smaller then $k^{|C|} \times (\lfloor L \rfloor + k)^{|C|}$, if we take the safe value $L = \max_{c \in C} \{L_c\}$.

Next, we want to prove that the grid and the corresponding TIOA display compatible behaviors, provided that the common input timed word is g-adjusted. But, before, we need to consider grid movements.

Definition 23 *Let M_G be a grid automaton. The* movement relation *of M_G, $\vdash_G$, is a binary relation over $\Sigma_G^\star \times S_G$ given by $(\sigma \cdot \psi, s_1) \vdash_G (\psi, s_2)$, with $\psi \in \Sigma_G^\star$ and $\sigma \in \Sigma_G$, if and only if there is a transition $(s_1, \sigma, s_2) \in T_G$.* •

The k-th power of $\vdash_G$ will be indicated by $\overset{k}{\vdash}_G$, $k \geq 0$, and its reflexive transitive closure by $\overset{\star}{\vdash}_G$. When $(\psi, \gamma) \overset{\star}{\vdash}_G (\varepsilon, \rho)$ we also write $\gamma \overset{\psi}{\Vdash}_G \rho$, or $\gamma \Vdash_G \rho$ when the particular input grid word is not relevant.

The next technical result will be useful later.

Lemma 24 *Let M be a TIOA and let M_G be the corresponding grid. Let L be a positive integer greater than all constants occurring in M, and let $g = 1/k$ with k a positive integer. Also, take $\mu \in [C \to \mathbb{Q}_{\geq}]$ and $p \in S$ with $(p, \mu_L) \in S_G$. Finally, assume that $\mu + \eta \vDash Inv(p)$, for some $i \geq 0$ and all $0 < \eta \leq ig$. Then $(p, \omega_L) \in S_G$ and $(p, \mu_L) \overset{g^i}{\Vdash}_G (p, \omega_L)$, with $\omega = \mu + ig$.*

Proof Recall that, over the grid alphabet, g^i represents a sequence of length i containing only g symbols. Define $\mu^j = \mu + jg$, for all j, $0 \leq j \leq i$. It suffices to prove that[3] $(p, \mu_L^j) \in S_G$ and $(p, \mu_L) \overset{g^j}{\Vdash}_G (p, \mu_L^j)$, for all j, $0 \leq j \leq i$. Note that $\mu_L^i = (\mu + ig)_L = \omega_L$. We proceed by induction on $j \geq 0$.

BASIS: when $j = 0$ we get $g^j = \varepsilon$. Then, trivially, $(p, \mu_L) \overset{g^j}{\Vdash}_G (p, \mu_L)$. We show $\mu_L = \mu_L^j$. We have $\mu^j = \mu + 0g = \mu + 0 = \mu$ and so $\mu_L^j = \mu_L$, as desired.

INDUCTION STEP: assume the result holds for some j, $0 \leq j < i$.

The induction hypothesis gives $(p, \mu_L) \overset{g^j}{\Vdash}_G (p, \mu_L^j)$ and $(p, \mu_L^j) \in S_G$. Note that $(p, \mu_L^j) \in S_G$ gives $(p, \mu_L^j) \in HS$ at line 23 of Algorithm 2. Also, pairs are moved from RS into HS one at a time at line 9. Hence, at some iteration, (p, μ_L^j) was chosen at line 8. We show that now line 17 applies.

Let $0 < \eta \leq g$. We show that $\mu_L^j + \eta \vDash Inv(p)$. We have $0 < jg + \eta \leq (j+1)g \leq ig$. From the hypothesis we get $\mu + (jg + \eta) \vDash Inv(p)$, that is $(\mu + jg) + \eta \vDash Inv(p)$,

[3] Here, μ_L^j denotes $(\mu^j)_L$.

and so $\mu^j + \eta \vDash Inv(p)$. Using Lemma 13 we get $\mu_L^j + \eta \vDash Inv(p)$, as desired. Thus, Algorithm 2, lines 18–20, will put $((p, \mu_L^j), g, (p, \rho_L))$ in T_G, with $\rho = \mu_L^j + g$. So, $(p, \mu_L^j) \overset{g}{\Vdash}_G (p, \rho_L)$. Also, by line 20, (p, ρ_L) will be added to RS if it is not already in HS. In any case, when the loop at line 7 terminates, we get (p, ρ_L) in HS and, by line 23, $(p, \rho_L) \in S_G$. Moreover, since $(p, \mu_L) \overset{g^j}{\Vdash}_G (p, \mu_L^j)$, we also get $(p, \mu_L) \overset{g^{j+1}}{\Vdash}_G (p, \rho_L)$. We extend the induction by showing that $\rho_L = \mu_L^{j+1}$. Since $\rho_L = ((\mu^j)_L + g)_L$, using Proposition 12 we get $\rho_L = (\mu^j + g)_L = (\mu + jg + g)_L = (\mu + (j+1)g)_L = (\mu^{j+1})_L = \mu_L^{j+1}$, as desired. •

Next, we want to guarantee that g-reachable configurations in a TIOA are states in the corresponding grid.

Lemma 25 *Let M_G be a grid corresponding to a TIOA M and let L be a positive integer greater than any constant occurring in M. If $(s, \nu) \in S \times [C \to \mathbb{Q}_{\geq}]$ is g-reachable in M then $(s, \nu_L) \in S_G$.*

Proof Using Definition 20, we know that $(s_0, \nu_0) \overset{\psi}{\Vdash}_M (s, \nu)$ for some g-adjusted timed word $\psi \in \Psi_{g,\Sigma}$. We proceed by induction on the length of ψ.

If $\psi = \varepsilon$ then $s_0 = s$ and $\nu_0 = \nu$. Algorithm 2, line 5, puts (s, ν) in RS. Now, the while loop at line 7, together with lines 8 and 9, will put (s_0, ν_0) in HS. Then, by line 23, we get $(s, \nu) \in S_G$.

Now, assume the result for any g-adjusted timed word ψ of length at most n, $n \geq 0$. Take $\varphi \in \Psi_{g,\Sigma}$ and $\sigma \in \Sigma \cup \mathbb{Q}_{\geq}$ with $\psi = \varphi \cdot \langle \sigma \rangle$, where φ has length n.

From $(s_0, \nu_0) \overset{\psi}{\Vdash}_M (s, \nu)$ we obtain $(s_0, \nu_0) \overset{\varphi}{\Vdash}_M (r, \mu)$ and $(r, \mu) \overset{\langle \sigma \rangle}{\Vdash}_M (s, \nu)$ for some $r \in S$ and $\mu \in [C \to \mathbb{Q}_{\geq}]$. Since ψ is g-adjusted, we have that φ is g-adjusted. By the induction hypothesis, we get $(r, \mu_L) \in S_G$. Then, $(r, \mu_L) \in HS$ (line 23). Clearly, from line 6 of Algorithm 2, together with the while loop at line 7 and lines 8 and 9, we can conclude that (r, μ_L) will be in RS. So, at some point, (r, μ_L) will be chosen at line 8.

We have two cases:

CASE 1: $\sigma \in \Sigma$.

Since $(r, \mu) \overset{\langle \sigma \rangle}{\Vdash}_M (s, \nu)$ we must have in M a transition $(r, \sigma, \delta, \theta, s)$, for some $\delta \in \Phi_C$ and some $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$, where $\mu \vDash \delta$ and $\mu \oplus \theta \vDash Inv(s)$, with $\nu = \mu \oplus \theta$.

From Lemma 14, we obtain $\mu_L \oplus \theta \vDash Inv(s)$ and, from Lemma 13, we get $\mu_L \vDash \delta$. Since (r, μ_L) will be chosen at line 8, from lines 12–14, the state (s, η), with $\eta = (\mu_L \oplus \theta)_L$, will be put into RS, if it is not already in HS. In any case, by the loop at line 7, we will get (s, η) in HS, and so by line 23 we will have $(s, \eta) \in S_G$. But, using Proposition 12, we

have $\eta = (\mu_L \oplus \theta)_L = (\mu \oplus \theta)_L$. Since $\nu = \mu \oplus \theta$, we obtain $\eta = \nu_L$ and so $(s, \nu_L) \in S_G$, as desired.

CASE 2: $\sigma \in \mathbb{Q}_{\geq}$.

Since ψ is g-adjusted, σ is also g-adjusted. Hence, $\sigma = kg$, for some $k \geq 0$. From $(r, \mu) \overset{\langle kg \rangle}{\Vdash_M} (s, \nu)$ we get $s = r$, $\nu = \mu + kg$ and $\mu + \eta \vDash Inv(r)$, for all $0 < \eta \leq kg$. Since we already have $(r, \mu_L) \in S_G$, Lemma 24 gives $(r, \nu_L) \in S_G$, completing the proof.

•

Conversely, we show that all grid states correspond to g-reachable configurations in the corresponding TIOA.

Lemma 26 *Let M_G be the grid corresponding to a TIOA M. Let L be a positive integer greater than any constant occurring in M. If $(s, \nu) \in S_G$ then there is a $\rho \in [C \to \mathbb{Q}_{\geq}]$ and a g-adjusted timed word $\psi \in \Psi_{g,\Sigma}$ such that $(s_0, \nu_0) \overset{\psi}{\Vdash_M} (s, \rho)$ and $\rho_L = \nu$.*

Proof From line 23 of Algorithm 2, we know that S_G is the set HS when the loop at line 7 terminates. From line 6, HS starts empty and elements are added to it one at a time and only at lines 14 and 20. Hence, it suffices to show that the result holds for all pairs (s, ν) added to RS at these lines.

Let (r_i, μ_i), $i \geq 0$, be the elements added to RS, in order. Clearly, $(r_0, \mu_0) = (s_0, \nu_0)$ by line 5. Taking $\psi = \varepsilon$, the result is seen to hold for (r_0, μ_0). Note also that $\nu_0 = (\nu_0)_L$, since $\nu_0(c) = 0$, for all $c \in C$.

Assume the result holds for (r_j, μ_j), for all $0 \leq j < k$, for some $k \geq 1$. Consider (r_k, μ_k). Since $k \geq 1$, (r_k, μ_k) was added to RS at line 14 or at line 20. Hence, at that iteration, some (r_j, μ_j) with $j < k$ was chosen at line 8. The induction hypothesis gives some $\psi \in \Psi_{g,\Sigma}$ such that $(s_0, \nu_0) \overset{\psi}{\Vdash_M} (r_j, \rho)$ and $\rho_L = \mu_j$. There are two cases.

CASE 1: At line 14. Then, from lines 10 and 11, we get a transition $(r_j, z, \delta, \theta, r_k)$ in T with $\mu_j \vDash \delta$ and $\mu_j \oplus \theta \vDash Inv(r_k)$. From lines 12 and 14, $\mu_k = (\mu_j \oplus \theta)_L$.

Since $\rho_L = \mu_j$, we get $\rho_L \vDash \delta$ and $\rho_L \oplus \theta \vDash Inv(r_k)$. From Lemma 13 we get $\rho \vDash \delta$ and from Lemma 13, with $\eta = 0$, we have $(\rho_L \oplus \theta)_L \vDash Inv(r_k)$. From Proposition 12 we may write $(\rho \oplus \theta)_L \vDash Inv(r_k)$ and so, from Lemma 13, with $\eta = 0$, we have $(\rho \oplus \theta) \vDash Inv(r_k)$.

Collecting, we have $(r_j, z, \delta, \theta, r_k)$ in T, $\rho \vDash \delta$ and $(\rho \oplus \theta) \vDash Inv(r_k)$. Then $(r_j, \rho) \overset{z}{\Vdash_M} (r_k, \rho \oplus \theta)$. Therefore, $(s_0, \nu_0) \overset{\psi \cdot \langle z \rangle}{\Vdash_M} (r_k, \rho \oplus \theta)$.

We complete this case by showing $(\rho \oplus \theta)_L = \mu_k$. From Proposition 12, $(\rho \oplus \theta)_L = (\rho_L \oplus \theta)_L$. Since $\rho_L = \mu_j$ and $\mu_k = (\mu_j \oplus \theta)_L$, we get $(\rho \oplus \theta)_L = (\mu_j \oplus \theta)_L = \mu_k$, as desired.

CASE 2: At line 20. Then from line 17 we obtain $\mu_j + \eta \vDash Inv(r_j)$, $0 < \eta \leq g$. And from lines 18 and 20 we get $\mu_k = (\mu_j + g)_L$ and $r_k = r_j$, respectively. Since $\mu_j = \rho_L$ we get $\rho_L + \eta \vDash Inv(r_j)$ and so $\rho + \eta \vDash Inv(r_j)$ by Lemma 13, for all $0 < \eta \leq g$. Then $(r_j, \rho) \overset{\langle g \rangle}{\Vdash_M} (r_j, \rho + g)$ and, since $r_j = r_k$ we get $(r_j, \rho) \overset{\langle g \rangle}{\Vdash_M} (r_k, \rho + g)$. Therefore, $(s_0, \nu_0) \overset{\psi \cdot \langle g \rangle}{\Vdash_M} (r_k, \rho + g)$.

We complete this case by showing that $(\rho + g)_L = \mu_k$. Since $\mu_k = (\mu_j + g)_L$ and $\rho_L = \mu_j$, we get $\mu_k = (\rho_L + g)_L$. Using Proposition 12, we conclude that $\mu_k = (\rho + g)_L$, as desired.

The induction is extended and we are done. •

The next definitions map adjusted timed words into grid words, and vice-versa.

Definition 27 *Let Σ be an alphabet and let $\psi \in \Psi_{g,\Sigma}$ be a g-adjusted timed word. The mapping $h : \Psi_{g,\Sigma} \to \Sigma_G^\star$ is defined by letting $h(\varepsilon) = \varepsilon$, and for all $\psi \in \Psi_{g,\Sigma}$ and all $\sigma \in \Sigma \cup \mathbb{Q}_{\geq}$:*

$$h(\psi \cdot \langle \sigma \rangle) = \begin{cases} h(\psi) \cdot \sigma & \text{if } \sigma \in \Sigma \\ h(\psi) \cdot g^k & \text{where } \sigma = kg, \text{ for some } k \geq 0. \end{cases} \bullet$$

Definition 28 *Let Σ be an alphabet and let $\psi \in \Sigma_G^\star$ be a grid word. The function $\widehat{h} : \Sigma_G^\star \to \Psi_{g,\Sigma}$ is defined by letting $\widehat{h}(\varepsilon) = \varepsilon$ and, for all $\varphi \in \Sigma_G^\star$ and all $\sigma \in \Sigma_G$, $\widehat{h}(\varphi \cdot \sigma) = \widehat{h}(\varphi) \cdot \langle \sigma \rangle$.* •

We note that $\widehat{h}(\psi)$ is also g-adjusted.

Proceeding, we show next that a grid automaton can homomorphically simulate the corresponding TIOA over a g-adjusted timed word.

Theorem 29 *Let M_G be the grid corresponding to a TIOA M. Let $s, r \in S$ and $\nu, \omega \in [C \to \mathbb{Q}_{\geq}]$ be such that (s, ν) is g-reachable in M with $g = 1/k$ and k a positive integer. Also let $\psi \in \Psi_{g,\Sigma}$ be a g-adjusted timed word, and $L \in \mathbb{Q}_{\geq}$ be a positive integer greater than all constants occurring in M. If $(s, \nu) \overset{\psi}{\Vdash_M} (r, \omega)$ then $(s, \nu_L), (r, \omega_L) \in S_G$ and $(s, \nu_L) \overset{h(\psi)}{\underset{G}{\Vdash}} (r, \omega_L)$.*

Proof Since (s, ν) is g-reachable in M, we get a g-adjusted timed word ψ' such that $(s_0, \nu_0) \overset{\psi'}{\Vdash_M} (s, \nu)$. Then $(s_0, \nu_0) \overset{\psi' \cdot \psi}{\Vdash_M} (r, \omega)$. Since $\psi' \cdot \psi$ is also g-adjusted, (r, ω) is also g-reachable in M. Thus, by Lemma 25, we get $(s, \nu_L), (r, \omega_L) \in S_G$. It remains to show that $(s, \nu_L) \overset{h(\psi)}{\underset{G}{\Vdash}} (r, \omega_L)$.

We proceed by induction on the length $n \geq 0$ of ψ, noting that $(s, \nu) \overset{\psi}{\Vdash_M} (r, \omega)$.

BASIS: when $n = 0$, we get $\psi = \varepsilon$ and so $s = r$ and $\nu = \omega$. Thus $(s, \nu_L) = (r, \omega_L)$ and we get $(s, \nu_L) \overset{\varepsilon}{\Vdash}_G (r, \omega_L)$. Since $h(\psi) = \varepsilon$, the basis is complete.

INDUCTION STEP: assume the result holds for all g-adjusted timed words of length at most n.

Take $\psi = \varphi \cdot \langle\sigma\rangle$, where φ has length n, and $\sigma \in \Sigma \cup \mathbb{Q}_{\geq}$. Then, $(s, \nu) \overset{\psi}{\Vdash}_M (r, \omega)$ gives $(s, \nu) \overset{\varphi}{\Vdash}_M (p, \mu)$ and $(p, \mu) \overset{\langle\sigma\rangle}{\Vdash}_M (r, \omega)$.

By the induction hypothesis, $(p, \mu_L) \in S_G$ and $(s, \nu_L) \overset{h(\varphi)}{\Vdash}_G (p, \mu_L)$. Since, by definition, $h(\psi) = h(\varphi) \cdot h(\langle\sigma\rangle)$, it remains to show that $(p, \mu_L) \overset{h(\langle\sigma\rangle)}{\Vdash}_G (r, \omega_L)$.

There are two cases: when $\sigma \in \Sigma$ and when $\sigma \in \mathbb{Q}_{\geq}$.

CASE 1: $\sigma \in \Sigma$.

Since $(p, \mu) \overset{\langle\sigma\rangle}{\Vdash}_M (r, \omega)$, we must have a transition $(p, \sigma, \delta, \theta, r)$ in M, with $\mu \vDash \delta$, $\omega = \mu \oplus \theta$, and $\omega \vDash Inv(r)$. Recall that $(p, \mu_L) \in S_G$. Hence, at some point, Algorithm 2 has chosen (p, μ_L) at line 8. From $\mu \vDash \delta$, Lemma 13 gives $\mu_L \vDash \delta$. From $\mu \oplus \theta \vDash Inv(r)$ and Lemma 14 we get $\mu_L \oplus \theta \vDash Inv(r)$. Then Algorithm 2, lines 12 and 13, adds $((p, \mu_L), \sigma, (r, \rho_L))$ to T_G, where $\rho = \mu_L \oplus \theta$. Hence, $(p, \mu_L) \overset{h(\langle\sigma\rangle)}{\Vdash}_G (r, \rho_L)$, since $h(\langle\sigma\rangle) = \sigma$.

We complete this case by showing that $\rho_L = \omega_L$. Since $\rho_L = (\mu_L \oplus \theta)_L$, Proposition 12 gives $\rho_L = (\mu \oplus \theta)_L$. Then $\rho_L = \omega_L$ since $\omega = \mu \oplus \theta$.

CASE 2: $\sigma \in \mathbb{Q}_{\geq}$.

Since $\psi = \varphi \cdot \langle\sigma\rangle$ is g-adjusted, we have that σ is also g-adjusted. Let $\sigma = ig$, for some $i \geq 0$. Moreover, since $(p, \mu) \overset{\langle ig\rangle}{\Vdash}_M (r, \omega)$, we conclude that $p = r$, $\omega = \mu + ig$ and $\mu + \eta \vDash Inv(r)$, for all $0 < \eta \leq ig$. Since we already have $(p, \mu_L) \in S_G$, Lemma 24 gives $(p, \rho_L) \in S_G$ and $(p, \mu_L) \overset{g^i}{\Vdash}_G (p, \rho_L)$, with $\rho = \mu + ig = \omega$. But $h(\langle\sigma\rangle) = g^i$ and $p = r$. So, we have $(r, \omega_L) \in S_G$ and $(p, \mu_L) \overset{h(\langle\sigma\rangle)}{\Vdash}_G (r, \omega_L)$, completing the proof. •

In order to illustrate Theorem 29, consider the TIOA shown in Figure 2. We represent a configuration (s, ν) of the TIOA by (s, t_1, t_2), where $t_1 = \nu(c_1)$ and $t_2 = \nu(c_2)$. Then, its start configuration is $(s_0, 0, 0)$. Chose a granularity of $g = \frac{1}{2}$, and also choose $K = 3$ and $L = 6$. Take the g-adjusted timed word $\alpha = 2 on 4\ on 5\ off$.

Computing over the input timed word α, from Figure 2 and starting at $s_0, 0, 0)$, we get

$$(s_0, 0, 0) \underset{2}{\rightarrow} (s_0, 2, 2) \overset{on}{\rightarrow} (s_1, 0, 2) \underset{4}{\rightarrow} (s_1, 4, 6) \overset{on}{\rightarrow} (s_1, 0, 6) \underset{5}{\rightarrow} (s_1, 5, 11) \rightarrow (s_0, 0, 0)\mathit{off}.$$

The corresponding partial grid is depicted in Figure 3, and the morphism h gives the grid word

$$h(\alpha) = \beta = (\frac{1}{2})^4\, on\, (\frac{1}{2})^8 on (\frac{1}{2})^{10} off.$$

Now we apply β to the grid, starting the run at the state $(s_0, 0, 0)$. From Figure 3, the grid moves as follows

$$(s_0,0,0) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_0,\tfrac{1}{2},\tfrac{1}{2}) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} \cdots \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_0,2,2) \overset{on}{\underset{G}{\Vdash}}$$

$$(s_1,0,2) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,\tfrac{1}{2},\tfrac{5}{2}) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} \cdots \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,4,6) \overset{on}{\underset{G}{\Vdash}} (s_1,0,6).$$

We continue following the grid movements in Figure 4:

$$(s_1,0,6) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,\tfrac{1}{2},\tfrac{13}{2}) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,1,6) \overset{\frac{1}{2}}{\underset{G}{\Vdash}}$$

$$(s_1,\tfrac{3}{2},\tfrac{13}{2}) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,2,6) \overset{\frac{1}{2}}{\underset{G}{\Vdash}}$$

$$\vdots$$

$$(s_1,\tfrac{9}{2},\tfrac{13}{2}) \overset{\frac{1}{2}}{\underset{G}{\Vdash}} (s_1,5,6) \overset{off}{\underset{G}{\Vdash}} (s_0,0,0).$$

Note that when, at state s_1, clock c_2 reaches the $\lfloor L \rfloor + 1 = 6 + 1 = 7$ boundary, then the corresponding grid state goes from $(s_1, t, 6)$ to $(s_1, t + \frac{1}{2}, \frac{13}{2})$, and back to $(s_1, t + 1, 6)$, and then cycles between 6 and $\frac{13}{2}$ for the value of clock 2. When the value of the first clock reaches 5, then a discrete movement is forced and both clocks are reset to zero.

The next theorem shows that a TIOA can also homomorphically imitate the corresponding grid automaton.

Theorem 30 *Let M_G be the grid corresponding to a TIOA M and let $g = 1/k$ with k a positive integer. Let $(s, \nu), (r, \omega) \in S_G$. Also let $L \in \mathbb{Q}_{\geq}$ be a positive integer greater than all constants occurring in M. If $(s, \nu) \overset{\psi}{\underset{G}{\Vdash}} (r, \omega)$ for some $\psi \in \Sigma_G^\star$, then there are $\rho, \mu \in [C \to \mathbb{Q}_{\geq}]$ such that $(s, \rho) \overset{\widehat{h}(\psi)}{\Vdash_M} (r, \mu)$, with $\nu = \rho_L$ and $\omega = \mu_L$.*

Proof We proceed by induction on the length $n \geq 0$ of ψ.

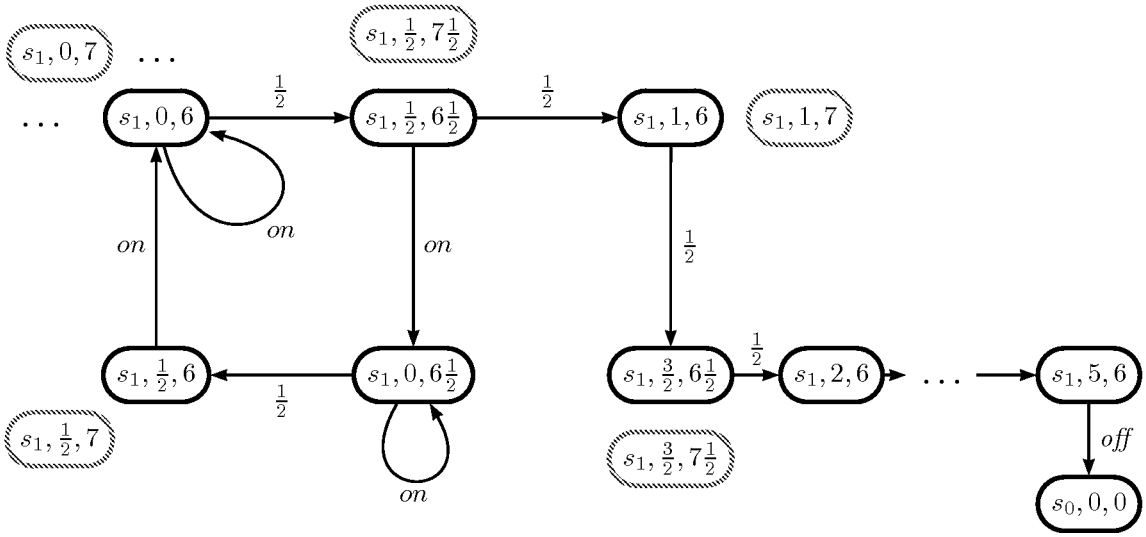


Figure 4: The partial grid automaton for the TIOA in Figure 2 (part 2 of 2).

BASIS: when $n = 0$, we get $\psi = \varepsilon$, $s = r$ and $\nu = \omega$. So, $\widehat{h}(\psi) = \varepsilon$. Using Lemma 26, we get $\rho \in [C \to \mathbb{Q}_{\geq}]$ with $\rho_L = \nu$. Take $\mu = \rho$. Then $(s, \rho) \overset{\widehat{h}(\psi)}{\Vdash_M} (r, \mu)$. Moreover, $\rho_L = \nu$ and $\mu_L = \rho_L = \nu = \omega$, completing the basis.

INDUCTION STEP: assume the result holds for all grid words of length at most n. Take $\psi = \varphi \cdot \sigma \in \Sigma_G^\star$, where φ has length n and $\sigma \in \Sigma_G$. Then, $(s, \nu) \overset{\psi}{\Vdash_G} (r, \omega)$ gives $(s, \nu) \overset{\varphi}{\Vdash_G} (p, \lambda)$ and $(p, \lambda) \overset{\sigma}{\Vdash_G} (r, \omega)$. By the induction hypothesis we get $\widehat{\nu}, \widehat{\lambda} \in [C \to \mathbb{Q}_{\geq}]$, where $\widehat{\nu}_L = \nu$ and $\widehat{\lambda}_L = \lambda$, and such that $(s, \widehat{\nu}) \overset{\widehat{h}(\varphi)}{\Vdash_M} (p, \widehat{\lambda})$. Since $\widehat{h}(\psi) = \widehat{h}(\varphi) \cdot \widehat{h}(\sigma)$, it remains to show that $(p, \widehat{\lambda}) \overset{\widehat{h}(\sigma)}{\Vdash_M} (r, \widehat{\omega})$, for some $\widehat{\omega} \in [C \to \mathbb{Q}_{\geq}]$, where $\widehat{\omega}_L = \omega$.

There are two cases: when $\sigma \in \Sigma$ and when $\sigma = g$.

CASE 1: $\sigma \in \Sigma$.

Since $(p, \lambda) \overset{\sigma}{\Vdash_G} (r, \omega)$, we must have a transition $((p, \lambda), \sigma, (r, \omega))$ in T_G. From Algorithm 2, lines 10 to 14, we have $(p, \sigma, \delta, \theta, r)$ in T, for some $\delta \in \Phi_C$, $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$, and such that $\lambda \vDash \delta$, $\lambda \oplus \theta \vDash Inv(r)$ and $\omega = (\lambda \oplus \theta)_L$.

Since $\lambda = \widehat{\lambda}_L$, we get $\widehat{\lambda}_L \vDash \delta$. Using Lemma 13 we get $\widehat{\lambda} \vDash \delta$. Also $\widehat{\lambda}_L \oplus \theta \vDash Inv(r)$, since $\lambda \oplus \theta \vDash Inv(r)$. Now, from Lemma 13 (with $\eta = 0$) we get $(\widehat{\lambda}_L \oplus \theta)_L \vDash Inv(r)$.

But $(\widehat{\lambda}_L \oplus \theta)_L = (\widehat{\lambda} \oplus \theta)_L$, using Proposition 12. Then $(\widehat{\lambda} \oplus \theta)_L \vDash Inv(r)$, and so from Lemma 13 (with $\eta = 0$) we get $\widehat{\lambda} \oplus \theta \vDash Inv(r)$.

Collecting, we have $\widehat{\lambda} \vDash \delta$, $\widehat{\lambda} \oplus \theta \vDash Inv(r)$ and $(p, \sigma, \delta, \theta, r) \in T$. Then $(p, \widehat{\lambda}) \overset{\widehat{h}(\sigma)}{\Vdash_M} (r, \widehat{\lambda} \oplus \theta)$, since $\widehat{h}(\sigma) = \sigma$. Let $\widehat{\omega} = \widehat{\lambda} \oplus \theta$. To complete this case, we need $\widehat{\omega}_L = \omega$. But $\widehat{\omega}_L = (\widehat{\lambda} \oplus \theta)_L = (\widehat{\lambda}_L \oplus \theta)_L$, by Proposition 12. Since $\widehat{\lambda}_L = \lambda$ we get $\widehat{\omega}_L = (\lambda \oplus \theta)_L = \omega$, as desired.

Case 2: $\sigma = g$.

Since $(p, \lambda) \overset{g}{\Vdash_G} (r, \omega)$, we must have a transition $((p, \lambda), g, (r, \omega))$ in T_G. From Algorithm 2, lines 17– 20, we have $p = r$, $\omega = (\lambda + g)_L$, and $\lambda + \eta \vDash Inv(p)$, for all $0 < \eta \leq g$.

Fix any η, $0 < \eta \leq g$. Since $\lambda = \widehat{\lambda}_L$, we get $\widehat{\lambda}_L + \eta \vDash Inv(p)$. Using Lemma 13 we get $(\widehat{\lambda}_L + \eta)_L \vDash Inv(p)$, and so $(\widehat{\lambda} + \eta)_L \vDash Inv(p)$, using Proposition 12. Hence $\widehat{\lambda}+\eta \vDash Inv(p)$, using Lemma 13 again. Therefore, $\widehat{\lambda}+\eta \vDash Inv(p)$, $0 < \eta \leq g$. Then $(p, \widehat{\lambda}) \overset{\langle g \rangle}{\Vdash_M} (p, \widehat{\lambda} + g)$. Since $p = r$, by letting $\widehat{\omega} = \widehat{\lambda} + g$ we get $(p, \widehat{\lambda}) \overset{\widehat{h}(\sigma)}{\Vdash_M} (r, \widehat{\omega})$, because $\widehat{h}(\sigma) = \widehat{h}(g) = \langle g \rangle$.

In order to complete this case, we need $\widehat{\omega}_L = \omega$. But $\widehat{\omega}_L = (\widehat{\lambda} + g)_L = (\widehat{\lambda}_L + g)_L$ by Proposition 12. Since $\widehat{\lambda}_L = \lambda$, we get $\widehat{\omega}_L = (\lambda + g)_L = \omega$, as desired.

This concludes the proof. •

4 Generating test cases based on test purposes

In this section we use the notion of a test purpose [20] in order to generate test sequences for TIOA. We also use the product of TIOA to capture the coordination between specifications of real-time systems and test purposes. The latter models specific properties to be verified about the original systems.

4.1 Test purposes

We start by noticing that several types of faults, or desired properties, of practical interest can be modeled by a specific family of TIOA.

Definition 31 *A TIOA is acyclic iff its subjacent directed graph, defined by taking states as nodes and transitions as edges, is acyclic. A test purpose is an acyclic TIOA with two special sets of states: a set $F \subseteq S$, of fail states, and a set $D \subseteq S$, of desired states, with $F \cap D = \emptyset$.*

•

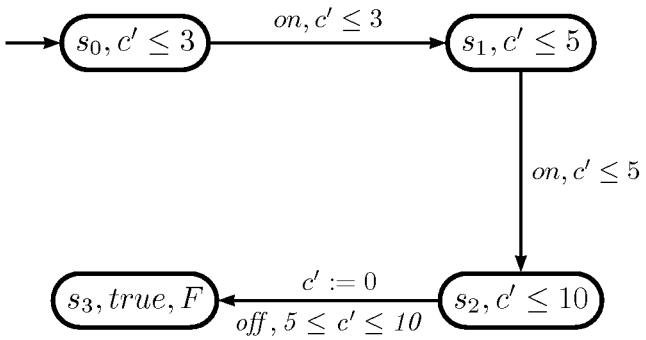


Figure 5: An example of test purpose for the switch system.

A test purpose focuses on specific parts of the system, with the aim of verifying whether the implementation meets certain properties. The fail states in F represent undesired, or fail, properties of the system. Later, a test purpose can be combined with the specification TIOA so that both are driven synchronously. Then, using the test purpose as a guide, one can extract certain input test sequences that drive the specification and the test purpose to combined fail states. Such a set of test sequences can then be applied to implementations in order to determine if they also reach a fail condition. A similar reasoning applies to the desired states in D.

As an example, the acyclic TIOA depicted in Figure 5 is related to the specification TIOA shown in Figure 1. Its intention is to verify whether the implementation accepts the input *on* within three time units followed by another *on* symbol which arrives within five time units, counting from the start. Then, the implementation must respond with an *off* symbol within no more than ten time units from the start, and no less than five time units also from the start. In this case, the special sets are $F = \{s_3\}$ and $D = \emptyset$.

Using test purposes one can focus on smaller parts of the specification model, thus avoiding generating test cases for the whole system. Together with a test purpose, verifying an implementation requires that it satisfies both the specification model and the test purpose. This will be formalized by constructing the (synchronous) product [21, 9] of the specification and the test purpose. Next, in order to control the number of states, we obtain the grid automaton associated with the product [9]. Then, we can traverse the resulting grid searching down for faulty or desired states, collecting along a set of input sequences. Test sequences are then derived from these input sequences. The test sequences can, then, be used to test implementation candidates in order to identify desired behaviors or faults, when those have been modeled by the test purpose.

4.2 The synchronous product

The intended meaning of a product of two TIOA is to force the synchronism between the participating TIOA, whenever possible. For that, states in the product are modeled as pairs of states from the participating TIOA. When a state (s_1, s_2) is in the product and there are transitions out of s_1 and of s_2 on a same action symbol a, then these transitions must be taken in parallel. In the product, this is modeled by inserting a transition out of (s_1, s_2), over the same symbol a, and in such a way as to capture the effects of the individual transitions in the participating TIOA. When only one of the states, s_1 or s_2, has an outgoing transition on a symbol a, then the product reflects that action, while keeping the other state unchanged. Clearly, the initial state in the product is the pair of initial states in the participating TIOA. Finally, in order to avoid clock reset conflicts, we require that both sets of clocks in the participating TIOA to be disjoint.

The synchronous product is constructed algorithmically.

Definition 32 *Let M_1 and M_2 be two TIOA, with $C_1 \cap C_2 = \emptyset$. The synchronous product of M_1 and M_2 is the TIOA $M_1 \otimes M_2$ constructed by Algorithm 3. If M_2 is a test purpose, then a state (s_1, s_2) in the product is a desired or fail state iff s_2 is a desired or fail state, respectively, in M_2.*

Algorithm 3 constructs the synchronous product for two TIOA by first pairing the initial state of both participating TIOA in order to create the initial state of the product TIOA. Then, the product transitions are constructed by an exhaustive search for new transitions and new states in the product. We want a state (s_1, s_2) to be a fail state in the product when s_2 is a fail state in the purpose model. Similarly for the desired states.

Consider the specification given in Figure 1 and the test purpose given in Figure 5. The resulting product is shown in Figure 6. The invariant conditions are shown in Table 1. Note that (q_0, s_3, F) and (q_1, s_3, F) are the fail states in the product automaton. Also, note that some input words that are accepted by the specification may not be accepted by the product, since particular timing requirements in the specification may not be satisfied simultaneously by the test purpose. For instance, taking $K = 5$, the timed word "$\frac{14}{3} on$" induces a run in the specification but not in the synchronous product since the invariant at (q_0, s_0) requires $c' \leq 3$ and we have $14/3 > 3$.

4.3 Test case generation and the grid automaton

We can generate timed test cases using the notion of a grid automaton extracted from the product of the specification and the test purpose, given by their respective TIOA.

The grid automaton is obtained from the resulting product following the method presented in Subsection 3.2. Then, test sequences are extracted by traversing the grid. The traversal operation starts at the initial state of the grid and searches down until it finds fail or desired

Input: TIOA M_1 and M_2 with $C_1 \cap C_2 = \emptyset$.
Output: The synchronous product $M_P = M_1 \otimes M_2$.
begin
  $C_P \leftarrow C_1 \cup C_2$ // the set of clock variables;
  $X_P \leftarrow X_1 \cup X_2$, $Y_P \leftarrow Y_1 \cup Y_2$, $\Sigma_P \leftarrow \Sigma_1 \cup \Sigma_2$ // action symbols;
  $RS \leftarrow s_0^P = (s_0^1, s_0^2)$, $Inv_P(s_0^P) \leftarrow Inv_1(s_0^1) \wedge Inv_2(s_0^2)$ // initial state;
  $T_P \leftarrow \emptyset$, $HS \leftarrow \emptyset$ // transitions and visited states;
  while $RS \neq \emptyset$ **do**
    Choose $s = (s_1, s_2)$ from RS;
    Move s from RS to HS;
    foreach $a \in \Sigma$ **do**
      if $(s_i, a, \delta_i, \theta_i, s_{i+2}) \in T_i$ *for some* $s_{i+1} \in S$, $\delta_i \in \Phi_{C_i}$, $\theta_i \in [C_i \curvearrowright \mathbb{Q}_{\geq}]$, $i = 1, 2$ **then**
        let $(p, q) = (s_3, s_4)$, $\delta = \delta_1 \wedge \delta_2$, $\theta = \theta_1 \oplus \theta_2$, $I = Inv_1(s_3) \wedge Inv_2(s_4)$
      end
      if $(s_1, a, \delta_1, \theta_1, s_3) \in T_1$ *for some* $s_3 \in S$, $\delta_1 \in \Phi_{C_1}$, $\theta_1 \in [C_1 \curvearrowright \mathbb{Q}_{\geq}]$ *and* $(s_2, a, \delta_2, \theta_2, s_4) \notin T_2$ *for all* $s_4 \in S$, $\delta_2 \in \Phi_{C_2}$,$\theta_2 \in [C_2 \curvearrowright \mathbb{Q}_{\geq}]$ **then**
        let $(p, q) = (s_3, s_2)$, $\delta = \delta_1$, $\theta = \theta_1$, $I = Inv_1(s_3) \wedge Inv_2(s_2)$
      end
      if $(s_2, a, \delta_2, \theta_2, s_4) \in T_2$ *for some* $s_4 \in S$, $\delta_2 \in \Phi_{C_2}$, $\theta_2 \in [C_2 \curvearrowright \mathbb{Q}_{\geq}]$ *and* $(s_1, a, \delta_1, \theta_1, s_3) \notin T_1$ *for all* $s_3 \in S$, $\delta_1 \in \Phi_{C_1}$,$\theta_1 \in [C_1 \curvearrowright \mathbb{Q}_{\geq}]$ **then**
        let $(p, q) = (s_1, s_4)$, $\delta = \delta_2$, $\theta = \theta_2$, $I = Inv_1(s_1) \wedge Inv_2(s_4)$
      end
      Add (p, q) to RS, **let** $Inv_P(p, q) \leftarrow I$, **add** $((s_1, s_2), a, \delta, \theta, (p, q))$ to T_P;
    end
  end
  $S_P \leftarrow HS$;
end

Algorithm 3: Synchronous product.

Table 1: Invariants for the product

State	Invariant
(q_0, s_0)	$(c' \leq 3) \wedge (c \leq K)$
(q_1, s_1)	$(c' \leq 5) \wedge (c \leq 5)$
(q_1, s_2)	$(c' \leq 10) \wedge (c \leq 5)$
(q_0, s_3)	$(c \leq K)$
(q_0, s_1)	$(c' \leq 5) \wedge (c \leq K)$
(q_1, s_3)	$(c \leq 5)$

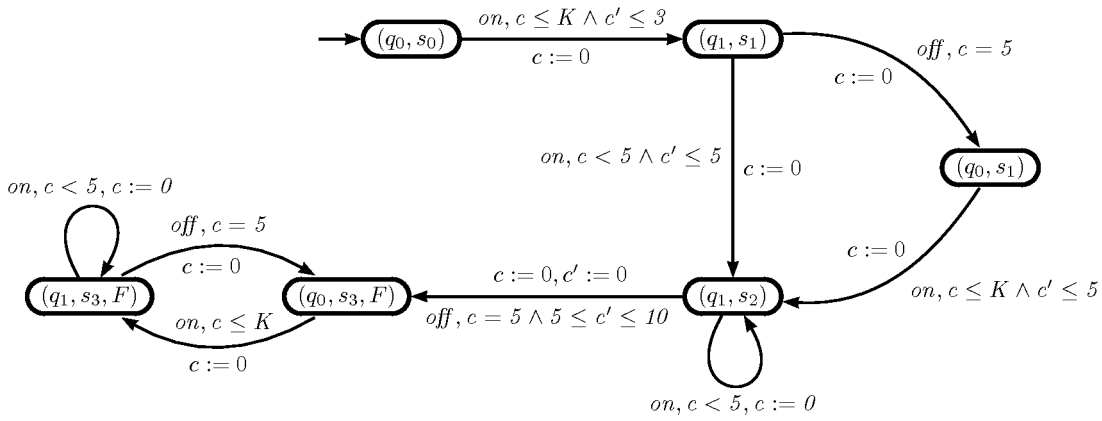


Figure 6: The product of a specification and a test purpose.

states, depending on the nature of the test. Upon finding one such state, the corresponding test sequence is output. A recursive traversal procedure is depicted in Algorithm 4. Clearly, the set of all test sequences can be generated by a call in the form $TestGeneration(s_0, \epsilon)$, where s_0 is the initial state of the grid obtained from the product automaton and ϵ is the empty string.

```
1  TestGeneration (INPUT: state s of a grid M_G; OUTPUT: A timed test sequence
   TTS;)
2  begin
3      if s is a leaf then
4          Write TTS;
5      else
6          foreach neighbor, r, of s reached over a transition on z do
7              Concatenate z with TTS;
8              TestGeneration (r,TTS);
9          end
10     end
11 end
```

Algorithm 4: The traversal algorithm *TestGeneration*.

Note that one can construct g-adjusted timed words from all grid words extracted from the grid automaton, using the mappings given at Definition 27 and Definition 28. Note also that delay transitions in the grid represent the continuous evolution in the original specification.

When the test purpose models desired behaviors, the verification process issues a “pass”

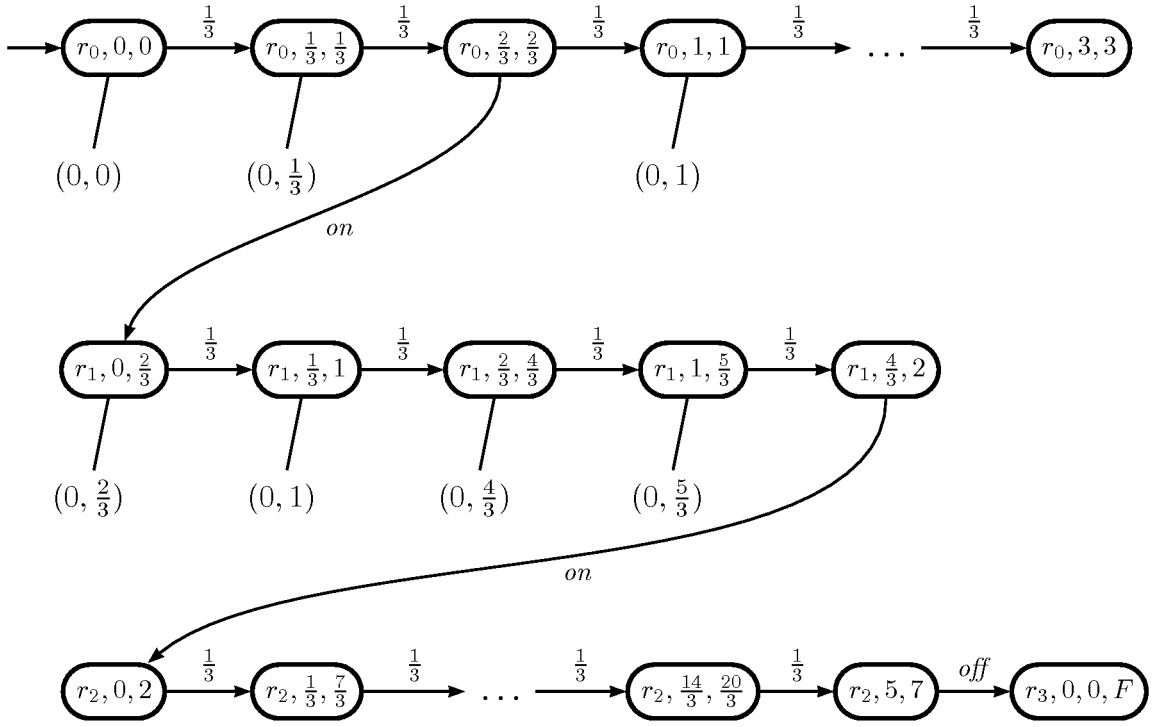


Figure 7: The partial grid automaton for Figure 6.

verdict only when the implementation is shown to respect the specification and satisfy the test purpose, for all sequences in the test suite. If, on the other hand, the testing if based on a purpose automaton with fail states, then the verification process issues a positive verdict only when the implementation satisfies the specification and also reaches a faulty state, for any of the sequences in the test suite.

Revisiting the last example, on page 60, and taking with $K = 5$, we see that the sequence $\beta = \frac{8}{3} on \frac{7}{3} on 5 off$ is an execution at the product shown in Figure 6. If an implementation under test conforms to the specification and the test purpose, we should be able to observe the same behavior when β is applied in this implementation. In Figure 7 we present a partial grid automaton extracted from the synchronous product shown in Figure 6, and where we have chosen a granularity of $\frac{1}{3}$. Note that we have relabeled the states in the figure in order to keep the notation uncluttered. Next, we can traverse the grid and extract grid words, as previously indicated by Algorithm 4. For example, the sequence $\alpha = (\frac{1}{3})^2 on (\frac{1}{3})^4 on (\frac{1}{3})^{15} off$ can be so extracted. Using the reverse mapping of Definition 28 we obtain the product timed word $\frac{2}{3} on \frac{4}{3} on 5 off$. Some other sequences that would be extracted from the grid and then mapped to timed words are:

$$\frac{1}{3}on\frac{14}{3}on5off \qquad \frac{2}{3}on\frac{13}{3}on5off \qquad 1on4on5off$$

$$\frac{4}{3}on\frac{11}{3}on5off \qquad \frac{5}{3}on\frac{10}{3}on5off \qquad 2on3on5off$$

$$\frac{7}{3}on\frac{8}{3}on5off \qquad \frac{8}{3}on\frac{7}{3}on5off \qquad 3on2on5off.$$

Note that some sequences cannot be obtained by traversing the grid, although they might be executions over the specification. For example, sequences $onon4off$ and $\frac{2}{3}on\frac{13}{3}on\frac{17}{3}off$ cannot be extracted from the grid of Figure 6. The first one has a total elapsed time of less than five time units after the last on and this is not allowed in the test purpose although it is a valid execution in the specification model. In the second sequence, the last off symbol occurs more than five time units after the previous on symbol, which is not allowed in the specification, even though it is allowed in the test purpose.

In the testing process we supply input actions to reactive real-time systems and observe its outputs. Then in order to drive a TIOA when testing properties, we need to supply it with a sequence of timed input symbols. In order to extract the behavior of a run, we need to project a timed word onto a subset of timed output actions.

Definition 33 *Let Σ be an alphabet and let $\Upsilon \subseteq \Sigma$. Let $\psi = \langle \sigma_1, \ldots, \sigma_n \rangle$, $n \geq 0$, be a timed word over Σ. The projection of ψ over Υ, denoted by $\psi{\downarrow_\Upsilon}$, is the timed word over Υ given by the concatenation $\psi{\downarrow_\Upsilon} = \phi_1 \cdot \phi_2 \cdot \ldots \cdot \phi_n$, where*

$$\phi_i = \begin{cases} \langle \sigma_i \rangle & \text{if } \sigma_i \in \Psi_\Upsilon, \\ \varepsilon & \text{otherwise.} \end{cases}$$

A test sequence *for a TIOA M is a timed word over its input actions, that is, an element of Ψ_X.* •

That is, the projection extracts only the actions in the subset of interest, together with timing values. For example, $\psi = \langle 1, a, 2, b, 3, d, 4 \rangle$ is a timed word over $\{a, b, d\}$. Let $\Upsilon = \{a, d\} \subseteq \{a, b, d\}$. Then $\psi{\downarrow_\Upsilon} = \langle 1 \rangle \cdot \langle a \rangle \cdot \langle 2 \rangle \cdot \varepsilon \cdot \langle 3 \rangle \cdot \langle d \rangle \cdot \langle 4 \rangle = \langle 1, a, 2, 3, d, 4 \rangle$.

Test cases are timed words where only input action symbols can occur.

Definition 34 *A* test sequence *for a TIOA M is a timed word over its input actions, that is, an element of Ψ_X.* •

Figure 8 illustrates a framework for extracting timed test cases.

Given a test sequence ψ and given a start configuration γ, we can discover all timed words that have ψ as a projection over X, the set of input action symbols, and that produce a run starting at γ.

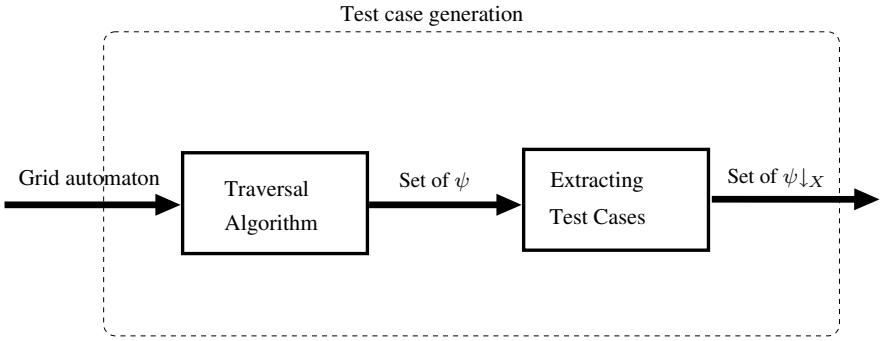


Figure 8: The framework to extract test cases.

Definition 35 *Let M be a TIOA and let $\psi \in \Psi_X$ and $\gamma \in \Gamma_M$. A* support *for ψ and γ is a timed word $\rho \in \Psi_M$ such that $\psi = \rho\downarrow_X$ and $\gamma \overset{\rho}{\Vdash} \mu$, for some $\mu \in \Gamma_M$.* •

That is, a support for an input timed word ψ and a configuration γ is a timed word ρ that drives the TIOA from γ to some configuration μ, and such that the projection of ρ over the set of input action symbols X is precisely ψ. The set of all supports for ψ and γ will be denoted by $\Lambda_{\psi,\gamma}$. When γ is the initial configuration of M, we also write Λ_ψ. Clearly, any $\rho \in \Lambda_\psi$ is a run of M.

We can now define observables associated with a test sequence and a start configuration.

Definition 36 *Let M be a TIOA and let $\psi \in \Psi_X$, $\gamma \in \Gamma_M$. The (*observable*)* behavior *of M from γ over ψ is the set $\{\rho\downarrow_X \mid \rho \in \Lambda_{\psi,\gamma}\}$.* •

We will denote by $\mathcal{O}_{\psi,\gamma}$ the set of observable behaviors of M from γ over ψ.

For example, consider the test case $\psi = \langle on, 5, L/2, on, 2\rangle$ for the TIOA depicted in Figure 1. We have the timed word, or test sequence, $\rho = \langle on, 5, \mathit{off}, K/2, on, 2\rangle$ with $\psi = \rho\downarrow_X$ and $(q_0, 0) \overset{\rho}{\Vdash} (q_1, 2)$, as it is easy to verify. Because $\rho\downarrow_Y = \langle 5, \mathit{off}, K/2, 2\rangle$, we put $\langle 5, \mathit{off}, L/2, 2\rangle \in \mathcal{O}_\psi$. It is clear that this is the only possibility for a timed word ρ with the properties that $(q_0, 0) \overset{\rho}{\Vdash} (s, t)$ and $\psi = \rho\downarrow_X$, for any state $s \in \{q_0, q_1\}$ and any clock value $t \in \mathbb{Q}_{\geq}$. We conclude that $\mathcal{O}_\psi = \{\langle 5, \mathit{off}, K/2, 2\rangle\}$.

4.4 Applying timed test cases

An implementation behavior is investigated by performing experiments over it [1]. Such experiments consist of applying stimuli to the implementation and observing its responses.

We obtain a set of timed test sequences by traversing the corresponding grid automaton which, in turn, was obtained from the product between the specification and the test purpose. Such test sequences are strings of input and output actions, as well as time delays. Timed test

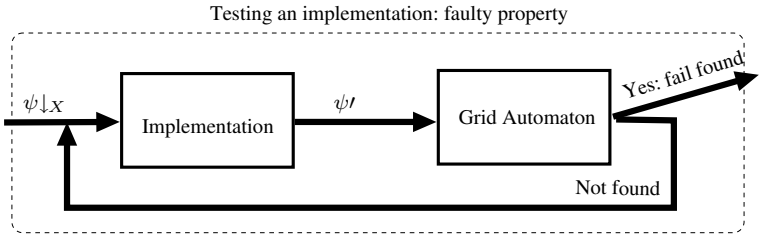


Figure 9: The framework to test implementations: faulty property.

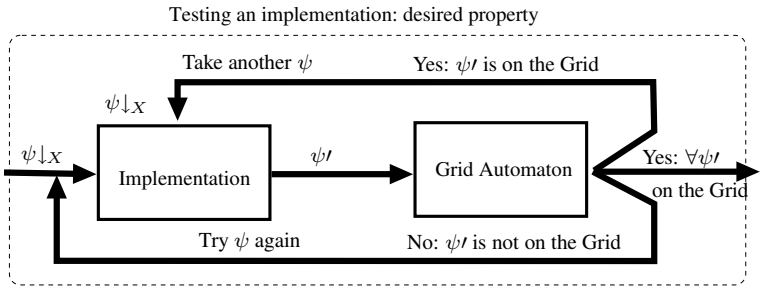


Figure 10: The framework to test implementations: desired property.

cases then are extracted from the timed test sequences, by projecting such sequences on the set of input action symbols. A test execution is obtained by applying a timed test case to an implementation and observing the respective responses. The output responses of the implementation under test are then combined with the respective timed test cases. By this process, we obtain test executions, also called *runs*, which combine input actions and time delays from the timed test cases, together with time delays and observed outputs from the implementation.

Suppose that a test case is submitted to an implementation under test. Then, usually, a time delay must pass before the next input action occurs, or before the next output action is observed. Note that the time delay preceding an output symbol is not under the control of the observer, and so we do not know in advance the exact instants when outputs will occur. In any case, when forming test run, the time delay actually observed before an output symbol occurrence is adjoined to the run being constructed, followed by the corresponding output symbol. When the next action is an input action the time delay specified on the test case is under control of the observer. In this case, this time delay, followed by the input symbol, is also adjoined to the run being collected.

Using this approach, a set of runs can be collected by applying timed test cases over the candidate implementation. We can, then, verify if runs obtained from the implementation candidate

are also runs over the corresponding grid automaton. When a run from the implementation is also a run over the grid, we can say that both models are in conformance, with respect to the given test case.

When a faulty property is specified by the test purpose model, the product automaton will contain special faulty states. The timed test cases extracted from the grid will be sequences of input actions and time delays that lead to faulty states. We then collect runs over the implementation using these timed test cases. By applying such runs back to the original grid, if any of them reaches a faulty state, we can announce a positive test verdict, that is, a fault was confirmed over the implementation. If all collected runs, when applied to the grid, do not reach faulty states, then the test is deemed inconclusive. See Figure 9. On the other hand, when the test purpose models a desired property, we must make sure that all runs, when applied to the grid, terminate at desired states. In this case, the test is positive, otherwise it is inconclusive. See Figure 10.

5 Related works

Many previous works have discussed formal methods to automatically generate test suites for timed systems. We briefly describe some such related works.

In [18], Springintveld et. al. present an exhaustive and general testing method for timed systems modeled by TIOA. They also use the notion of grid automata. However, this proposal demands that grid automata represent all clock regions defined by a finite set of clock interpretations in the TIOA. The classical notion of clock regions imposes this strict relationship between the number of clock variables present in the models and the granularity that must be chosen in order to obtain the corresponding grid automaton. A potential problem here is the very large number of test sequences that are generated after constructing grid automata.

In [9], En-Nouaary and Dssouli also discuss a model-based testing approach which uses test purposes and synchronous products. However, their discretization technique is also based on the classical notion of clock regions, thus imposing the same strict relationship between the number of clocks and the granularity that must be chosen in order to obtain the corresponding discrete automaton. Here, instead of constructing a grid automaton directly, the notion of a region graph is first used to represent a possibly infinite transition system and, then, by a process of sampling, a grid automaton is derived. However, it is not detailed how one could apply the timed test cases that are generated.

Another test generation method based on TIOA is proposed in [22]. Test purposes are also discussed following similar lines as in [9]. In these works, region graphs are also sampled in order to obtain grid automata. But no guarantees are postulated about the algorithms presented therein, which are used in an exhaustive test generation process. Also, dense time has but a superficial treatment, making it difficult to see how to generate precise timed test cases when

combining test purposes and TIOA models.

In [23], Fouchal and co-workers discuss a test execution strategy similar to the one discussed in this work. But, whereas our work deals with dense time in order to capture timed properties, in [23] the notion of timed elements are used to imitate continuous time evolution, thus offering no guarantees of accuracy about the extracted timed test sequences.

Another approach is taken in [24], where a conformance relation based on the ioco framework is proposed. However, there is no mechanism to discretize the continuous behavior. As a consequence, time evolution is not captured in appropriate manner for precisely testing timed properties.

In [25], a black-box conformance testing for real-time systems is presented. It uses a symbolic reachability graph to reduce the number of states and edges and, consequently, reducing the number of test sequences. The same ioco framework as used in [24] is applied, and so the same limitations for capturing continuous behavior of timed systems appear in this approach. The idea of choosing "ticks" to capture continuous behavior is similar to choosing granularities. However, in [25], there are no guarantees about homomorphic simulations between grid automata and the original TIOA, as given on our proposal.

Many similar approaches [26, 11, 6] use the classical notion of clock regions in order to obtain grid automata. In these cases, if a large number of clock variables is present in models, it will often lead to huge grid automata, due to the exponential number of clock regions that are obtained, and the need to enforce the relationship between the number of clocks and the chosen granularity. By contrast, our approach allows for an ample range of choice for appropriate granularity values, thus leading to more controllable grid automata and to potentially more manageable test suites.

6 Concluding Remarks

Many approaches have been proposed to automatically construct test suites for timed systems. The problem is specially challenging since timed systems exhibit continuous time evolution and, often, formal models techniques for dealing with such systems incur in the known state explosion problem.

In this work we proposed a new way of discretizing timed system models. In the process, we noticed that the classical notion of clock regions were unnecessary, giving rise to a more general notion that allowed for an ample range of granularity values that could be chosen in the discretization process. Since the specification is well-known by the tester, using this approach we can choose adequate granularities to construct suitable grids, thus exercising more control over the state space explosion problem without loss of accuracy. We also demonstrated that the grid automaton obtained using our method was capable of homomorphically simulating the original timed system, and vice-versa. This formed the basis for the development of automatic

methods for generating test suites.

In order to test more specific system properties over the implementations, we made use of the notion of timed test purpose models. Using the notion of a synchronous product between a timed system and a timed purpose model, the discretization algorithm is able to generate a grid automaton that reflects both the behavior of the original timed system, as well as the desired properties specified by the timed test purpose model. From this grid automaton, one can then automate the process of extracting test case sequences. Detailed proofs of correctness were provided for all properties of interest.

As suggestions for expanding this work, we cite the possible representation of data flow within the formal models, thus also capturing the idea of system context variables. With that notion in place, one should be able to deal both with control flow and time evolution, as well as with data flow issues, the latter being captured by context variables manipulations.

References

[1] Tretmans J. Model based testing with labelled transition systems. *Formal Methods and Testing*, 2008; 1–38.

[2] Cunning SJ, Rozenblit JW. Automating test generation for discrete event oriented embedded system s. *J. Intell. Robotics Syst.* 2005; **41**(2-3):87–112, doi:http://dx.doi.org/10.1007/s10846-005-3810-8.

[3] Nielsen B, Skou A. Test generation for time critical systems: Tool and case study. *ecrts* 2001; **00**:0155, doi:http://doi.ieeecomputersociety.org/10.1109/EMRTS.2001.934021.

[4] Tretmans J. Testing concurrent systems: A formal approach. *CONCUR '99: Proceedings of the 10th International Conference on Co ncurrency Theory*, *Lecture Notes in Computer Science*, vol. 1664, Baeten J, Mauw S (eds.), Springer-Verlag: London, UK, 1999; 46–65.

[5] Gawlick R, Segala R, Sogaard-Andersen JF, Lynch NA. Liveness in timed and untimed systems. *Automata, Languages and Programming*, 1994; 166–177. URL `citeseer.ist.psu.edu/article/gawlick94liveness.html`.

[6] En-Nouaary A, Dssouli R, Khendek F, Elqortobi A. Timed test cases generation based on state characterization technique. *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, IEEE Computer Society: Washington, DC, USA, 1998; 220.

[7] Alur R, Dill DL. A theory of timed automata. *Theoretical Computer Science* 1994; **126**(2):183–235. URL `citeseer.ist.psu.edu/alur94theory.html`.

[8] Alur R. Timed automata. *CAV'99*, no. 1633 in LNCS, 1999.

[9] En-Nouaary A, Dssouli R. A guided method for testing timed input output automata. *TestCom*, 2003; 211–225.

[10] Cardell-Oliver R. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing* 2000; **12**(5):350–371. URL `citeseer.ist.psu.edu/385816.html`.

[11] En-Nouaary A. A scalable method for testing real-time systems. *Software Quality Control* 2008; **16**(1):3–22, doi:http://dx.doi.org/10.1007/s11219-007-9021-8.

[12] Larsen KG, Yi W. Time abstracted bisimulation: Implicit specifications and decidability. *LNCs* 1994; **802**:160–176.

[13] Tretmans J. Test generation with inputs, outputs, and quiescence. *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, *Lecture Notes in Computer Science*, vol. 1055, Margaria T, Steffen B (eds.), Springer, 1996; 127–146.

[14] Gargantini A. Conformance testing. *Model-Based Testing of Reactive Systems: Advanced Lectures*, *Lecture Notes in Computer Science*, vol. 3472, Broy M, Jonsson B, Katoen JP, Leucker M, Pretschner A (eds.), Springer-Verlag, 2005; 87–111.

[15] Ghriga M, Frankl PG. Adaptive testing of non-deterministic communication protocols. *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, North-Holland Publishing Co.: Amsterdam, The Netherlands, The Netherlands, 1994; 347–362.

[16] Bonifácio AL, Moura AV. A New Timed Discretization Method for Automatic Test Generation for Timed Systems. *Technical Report IC-09-31*, Institute of Computing, University of Campinas September 2009.

[17] Bonifácio AL, Moura AV, Simão AS, Maldonado JC. Conformance Testing by Model Checking Timed Extended Finite State Machines. *Brazilian Symposium on Formal Methods (SBMF'06)*, Natal, 2006; 43–58.

[18] Springintveld J, Vaandrager F, D'Argenio PR. Testing timed automata. *Theor. Comput. Sci.* 2001; **254**(1-2):225–257.

[19] Wang F. Efficient verification of timed automata with bdd-like data structures. *STTT* 2004; **6**(1):77–97.

[20] Jeannet B, Jéron T, Rusu V. Model-based test selection for infinite-state reactive systems. *FMCO*, 2006; 47–69.

[21] Bonifácio AL, Moura AV, da Silva Simão A, Maldonado JC. Towards Deriving Test Sequences by Model Checking. *Electron. Notes Theor. Comput. Sci.* 2008; **195**:21–40, doi: http://dx.doi.org/10.1016/j.entcs.2007.08.025.

[22] Fouchal H. Conformance testing techniques for timed systems. *SOFSEM*, 2002; 1–19.

[23] Fouchal H, Petitjean E, Salva S. Testing timed systems with timed purposes. *RTCSA '00: Proceedings of the Seventh International Conference on Real-Time Systems and Applications*, IEEE Computer Society: Washington, DC, USA, 2000; 166.

[24] Briones LB, Brinksma E. A test generation framework for *uiescent* real-time systems. *FATES*, 2004; 64–78.

[25] Krichen M, Tripakis S. Conformance testing for real-time systems. *Form. Methods Syst. Des.* 2009; **34**(3):238–304, doi:http://dx.doi.org/10.1007/s10703-009-0065-1.

[26] En-Nouaary A, Dssouli R, Khendek F. Timed wp-method: Testing real-time systems. *IEEE Trans. Softw. Eng.* 2002; **28**(11):1023–1038, doi:http://dx.doi.org/10.1109/TSE.2002.1049402.

Epílogo

Este capítulo apresentou a proposta de uma nova discretização de modelos temporizados. Mostramos que não é necessário o uso das regiões de relógio para se obter uma discretização. Oferecemos uma flexibilidade na escolha da granularidade que por consequência proporciona uma diminuição significativa no espaço de estado gerado.

O objetivo do trabalho foi apresentar as provas de corretude da simulação entre o TIOA e seu correspondente autômato grid. A partir do modelo discretizado, sequências de teste podem ser extraídas do grid para o teste de conformidade. As propriedades modeladas pelas propostas de teste a serem verificadas nos sistemas são classificadas como de falha ou desejada, de acordo com o foco do teste em questão. O critério de conformidade seguiu com uma estratégia de verificação usando sequências extraídas, tanto das aplicações realizadas nas implementações candidatas quanto nas realimentações feitas nos autômatos grid que representam os TIOA originais. Usando esta noção de teste mostramos que é possível verificar se o comportamento de ambos os modelos são compatíveis, produzindo assim vereditos sobre as implementações testadas com respeito as propriedades modeladas.

O próximo capítulo apresenta um modelo estendido de FSM, suportando variáveis de contexto, juntamente com um modelo de tempo, chamado TEFSM. O trabalho mostra como o problema de confirmação de configuração pode ser reduzido ao problema de se encontrar caminhos no produto de TEFSM, usando técnicas de model-checking para gerar seqüências de confirmação.

Capítulo 4

Derivando Sequências de Confirmação

Prólogo

A geração de casos de teste baseado em modelos tem sido largamente usada na atividade de teste de conformidade. Naturalmente em alguns casos, essa tarefa se torna altamente problemática devido à explosão no número de estados, especialmente considerando aspectos como as propriedades temporais de sistemas críticos.

Devido a essa complexidade, se faz necessário o uso de modelos formais e técnicas adequadas para manipular e contornar tal problema. Algumas técnicas de teste suportam a noção de variáveis de contexto em seus modelos tanto quanto parâmetros de entrada de saída, permitindo a representação de diferentes configurações do sistema. Já outras propostas incorporam a noção de tempo, permitindo que a evolução contínua de tempo dos sistemas seja capturada.

Neste trabalho, usamos um modelo estendido de FSM, suportando variáveis de contexto, juntamente com um modelo de tempo. Por isso, através dessa proposta podemos considerar interações entre entradas e saídas do modelo, bem como o instante em que esses eventos ocorrem. Neste caso, vale ressaltar que as entradas e saídas ocorrem de maneira associada.

Este capítulo é composto pelo artigo, "Towards Deriving Test Sequences by Model Checking", que explora o problema de confirmação de configurações no modelo TEFSM. A idéia é mostrar como o problema de confirmação de configuração pode ser reduzido ao problema de se encontrar caminhos no produto de TEFSM. Quando nenhum caminho é encontrado para uma propriedade especificada, técnicas de *model-checking* podem ser usadas para gerar uma seqüência de confirmação. Neste trabalho mostramos como usar a noção de modelos na geração de casos de teste, e como seqüências de confirmação podem ser aplicadas ao formalismo proposto. O artigo foi publicado na revista *Electronic Notes in Theoretical Computer Science* (ENTCS) da Elsevier Science Publishers. Uma versão preliminar deste artigo, intitulada "Conformance Testing by Model Checking Timed Extended Finite State Machines", foi apresentada e publicada no *Brazilian Symposium on Formal Methods* (SBMF 2006), realizado em Natal.

Towards Deriving Test Sequences by Model Checking

Adilson L. Bonifácio[1] Arnaldo V. Moura[2]
Computing Institute, University of Campinas
P.O. 6176 – Campinas – Brazil – 13081-970
Adenilso da Silva Simão[3] José Carlos Maldonado[4]
Mathematic Science and Computing Institute, University of São Paulo
P.O. 668 – São Carlos – Brazil – 13560-970

Abstract
Model-based testing automatically generates test cases from a model describing the behavior of the system under test. Although there exist several model-based formal testing methods, they usually do not address time constraints, mainly due to the fact that some supporting formalisms do not allow a suitable representation of time. In this paper, we consider such constraints in a framework of Timed Extended Finite State Machines (TEFSMs), which augment the Extended Finite State Machine (EFSM) model by including a notion of explicit and implicit time advancement. We use this extension to address conformance testing by reducing the confirming configuration problem to the problem of finding a path in a TEFSM product.

Keywords: Model Checking, Timed EFSM, Conformance testing, Suspicious Configuration.

1 Introduction

Model-based testing comprises the automatic generation of efficient test cases using models of system requirements, usually based on formally specified system functionalities. It involves the (i) construction of a suitable formal model, (ii) derivation of test inputs, (iii) calculation of test

[1] adilson@ic.unicamp.br
[2] arnaldo@ic.unicamp.br
[3] adenilso@icmc.usp.br
[4] jcmaldon@icmc.usp.br

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: {www.elsevier.nl/locate/entcs}

outputs, (iv) execution of test inputs over implementations, (v) comparison of the results from the calculated test outputs and the implementation executions, and (vi) decision of whether the testing should be stopped. All these tasks are tightly related to each other. For instance, the way the model is written impacts on how test inputs can be generated. Moreover, the decision of whether the implementation has already been tested enough depends on one's ability to determine how many undiscovered faults may remain in it. Usually the purpose of testing is not to demonstrate that the implementation is equivalent to its specification, since this goal is infeasible for most practical applications. Instead, this ideal equivalence is relaxed into a conformance relation [13, 15]. The so-called conformance testing aims at demonstrating that the implementation behavior conforms (in some sense) to the behavior dictated by the specification [29].

The problem of generating test cases for conformance testing based on Finite State Machines (FSMs) has already been investigated [7, 21, 28, 8, 14, 12]. However, there are many situations in which the modeling of the system as a FSM is cumbersome, due to the state explosion problem, or even impossible, due to the fact that there are some relevant aspects that can not be properly expressed, e.g., the passage of time. Some extensions to the FSM model have been proposed in order to overcome these problems [33, 6, 1]. Other extensions incorporate notions like context variables and input/output parameters, allowing the succinct representation of many different configurations [27]. Still others incorporate notions of time, allowing the model to capture the evolution of time [24, 4, 36].

An Extended Finite State Machine (EFSM) can be thought of as a folded FSM [27]. Given an EFSM, and assuming that domains are finite, it is possible to unfold it into a pure FSM by expanding the values of its parameters and variables. The resulting FSM can be used with FSM-based methods for test derivation with complete fault coverage, which means that all fault possibilities can be exhausted. Nonetheless, in most practical situations, this approach is unfeasible, mainly due to the state explosion effect [22, 27].

Time plays an important role in determining the acceptability of system behavior in many system categories since not only the input/output relationship can be relevant, but also the period of time when those events occur may be important. In such cases, it is mandatory to be able to represent time constraints of the system, and to test whether a given implementation conforms to these constraints. There are some formalisms that allow the representation of various time related concepts, such as Timed Petri Nets [19] and Timed Automata [2, 1, 32, 11]. Nonetheless, there are few, if any, methods that allow a satisfactory derivation of adequate test cases from those models.

We are interested in model-based methods for testing systems with time constraints. In particular, we are addressing the problem posed in tasks (i)-(iii) alluded to above, namely the construction of an adequate formalism for modeling systems and the automatic generation of test cases, as well as the determination of the expected outputs. These tasks are closely related,

and should be considered together.

To this end, we define Timed EFSMs [5], or TEFSMs, by including the notion of explicit and implicit time advancement in the EFSM formalism. Then, we can adapt some well-established results, derived for FSMs and EFSMs, to the context of systems that require time constraints. In particular, we address the problem of configuration confirmation for TEFSMs in the same vein as done by Petrenko et al. for EFSMs [27]. In that work, it is shown how the problem of configuration confirmation for EFSMs can be reduced to the problem of finding a path in an EFSM product. By defining a property that states when no such a path exists, model-checking techniques can be used to generate a confirming sequence. We show how the notion of product machines and confirming sequences can be applied to the extended formalism of TEFSMs. Given a configuration and a set of suspicious configurations, a confirming sequence is a sequence of (parameterized) inputs that allows us to distinguish the given configuration from suspicious configurations by comparing outputs and, possibly, observing the time indicated in each of the outputs. Finding a confirming sequence can also be seen as an extension of the state identification problem [20, 16].

This paper is organized as follows. In Section 2 we present the concepts of EFSMs and Extended Timed Transition Systems [7]. In Section 3 we introduce the Timed Extended FSMs. The product of TEFSMs is presented in Section 4. In Section 5 we describe how the TEFSM product can be used in a model-checking set-up, and illustrate this process with a simple example in Section 6. Finally, in Section 7, we draw some concluding remarks and indicate possible directions for future research.

2 Basic Formal Concepts

In this section, we give a brief overview of the formal concepts that are involved in this work. First, we present EFSMs which are used to specify system requirements. Next, important aspects of extended timed transition systems are introduced.

2.1 Extended FSM Model

An EFSM is an extension of a conventional FSM. In contrast to FSMs, in the EFSM model we have to consider other items [27], such as input and output parameters, and context variables. Also, update and output functions, as well as predicates are defined over context variables and input parameters.

Let X and Y be finite sets of input and output symbols. Let R be a finite set of parameter symbols. For $z \in X \cup Y$, we denote by $R_z \subseteq R$ the set of parameters associated with z. Also D_z denotes the set of parameter valuations associated with z. An element of D_z maps R_z to some valuation domain. Similarly, let V be a finite set of context variable names, with

D_V denoting a set of valuations for V. At this point, there is no need to further specify the valuation domains. An EFSM M over X, Y, R, V and the associated valuation domains is a tuple (S, T, s_0, λ_0), where S and T are finite sets of states and transitions, respectively, $s_0 \in S$ is the initial state, and λ_0 is an initial context variable valuation. Each transition $t \in T$ is a tuple (s, x, P, op, y, up, s'), where:

- $s, s' \in S$ are the source and the target states of the transition, respectively;
- $x \in X$ is the input symbol of the transition;
- $y \in Y$ is the output symbol of the transition;
- P, op and up are functions defined over valuations of the input parameters and context variables V, thus:
  - $P : D_x \times D_V \rightarrow \{True, False\}$ is the predicate of the transition;
  - $op : D_x \times D_V \rightarrow D_y$ is the output parameter function of the transition;
  - $up : D_x \times D_V \rightarrow D_V$ is the context update function of the transition.

Given an input x and the set of input parameter valuations D_x, a parameterized input is a pair (x, p_x), where $p_x \in D_x$. The parameterized outputs are defined in a similar way. A configuration of M is a pair $(s, \lambda) \in S \times D_V$, where s is a state and λ is a context variable valuation. A transition (s, x, P, op, y, up, s') is enabled for a configuration (s, λ) and parameterized input (x, p_x) if $P(p_x, \lambda)$ evaluates to true.

The machine starts from the initial configuration and operates as follows. Upon receiving an input along with the corresponding parameter valuation, and computes the predicates that are satisfied for the current configuration. From among the presently enabled transitions one will fire. By executing the chosen transition, the machine produces an output along with an output parameter valuation using of the output parameter function. The latter is computed by the output parameter valuation. The machine updates the current context variable valuation according to the context update function, and moves from the source to the target state of the transition.

An EFSM, furthermore, is considered to be:

- Predicate complete: for each pair $(s, x) \in S \times X$, every element in $D_x \times D_V$ evaluates at least one predicate to true among the set of all predicates guarding transitions leaving s with input x;
- Input complete: for each pair $(s, x) \in S \times X$, there exists at least one transition leaving state s with input x;

- Deterministic: any two transitions leaving the same state and with the same input have mutually exclusive predicates;
- Observable: for each state s and each input x, every outgoing transition from s on x has a distinct output symbol.

2.2 Extended Timed Transition Systems

We can extend the original *timed transition system* (TTS) notion of [7] by associating a set of clocks and invariant conditions with each state. All clocks in the model increase in an uniform way, according to a global time frame [1, 2], and the corresponding invariant condition must hold in the current state of the model.

First, we say how clocks behave during system evolution [1]. Let C be the set of clock names (or clocks, for short), $\Phi(C)$ is the set of clock constraints δ in the form,

$$\delta := c \leq \tau \mid \tau \leq c \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where c is a clock and $\tau \in \mathbb{Q}$[5] is a time instant. A clock interpretation, ν, is a mapping from C to $\mathbb{Q}$. The set of clock interpretations is denoted by $[C \mapsto \mathbb{Q}]$. An interpretation ν over C satisfies $\delta \in \Phi(C)$, written $\nu \vDash \delta$, iff δ evaluates to true when each clock c is substituted by $\nu(c)$ in δ.

Let $\nu \in [C \mapsto \mathbb{Q}]$ be a clock interpretation. For $\tau \in \mathbb{Q}$, we define the clock interpretation $\nu + \tau$, which maps each clock c to the value $\nu(c) + \tau$. Also, for $K \subseteq C$, $[K \mapsto \tau]\nu$ is the clock interpretation that assigns $\tau \in \mathbb{Q}$ to each clock $c \in K$ and agrees with ν on the rest of the clocks.

An Extended TTS (ETTS) is given by a tuple $(S, s_0, X, C, Inv, \longrightarrow)$, where S is a finite set of states, $s_0 \in S$ is the initial state, X is a finite set of events, C is a finite set of clocks, $Inv : S \rightarrow \Phi(C)$ maps states to invariant conditions, and $\longrightarrow$ is a transition relation, where $\longrightarrow \subseteq (S \times X \times 2^C \times \Phi(C) \times S)$. A configuration is given by a pair (s, ν), where s is a state and ν is a clock interpretation. The initial configuration is given by (s_0, ν_0), where $\nu_0(c) = 0$, for all $c \in C$, is the initial clock interpretation, and $\nu_0 \vDash Inv(s_0)$. Given a configuration (s, ν), a transition (s, x, K, δ, s') indicates that from state s, receiving the input event x, and provided that ν satisfies δ, the system may move to state s', resetting all the clocks in K to zero. The ETTS always starts in the initial configuration (s_0, ν_0), and with the (global) time set to zero.

A time sequence is a sequence $\bar{\tau} = \tau_0\tau_1\tau_2\ldots$, where $\tau_i \in \mathbb{Q}$, $i \geq 0$, $\tau_0 = 0$, and $\tau_i \geq \tau_{i-1}$, $i \geq 1$. A timed sequence is a pair $(\bar{x}, \bar{\tau})$, where $\bar{\tau}$ is a time sequence and $\bar{x} = x_0x_1x_2\ldots$ is a sequence of input symbols. The intuitive idea is that the symbol x_i occurs at time τ_i. Given two configurations, (s_1, ν_1) and (s_2, ν_2), a time delay $\tau \geq 0$ and an input x, we say that (s_2, ν_2)

[5] $\mathbb{Q}$ is the set of rationals and $\mathbb{Q}^{>0}$ is the set of positive rationals.

evolves from (s_1, ν_1) over τ and x, denoted by $(s_1, \nu_1) \xrightarrow[\tau]{x} (s_2, \nu_2)$, iff there is a transition (s_1, x, K, δ, s_2) such that:

1. $\nu_1 + \eta \vDash Inv(s_1)$ for all $0 \leq \eta \leq \tau$,
2. $\nu_1 + \tau \vDash \delta$,
3. $\nu_2 = [K \mapsto 0](\nu_1 + \tau)$, and
4. $\nu_2 \vDash Inv(s_2)$.

A sequence of configurations $\bar{\gamma} = \gamma_0\gamma_1\gamma_2\ldots$ is a run of M iff γ_0 is the initial configuration of M, and there is a timed input $(\bar{x}, \bar{\tau})$ such that

$$\gamma_{i-1} \xrightarrow[\theta_i]{x_i} \gamma_i, \text{ where } \theta_i = \tau_i - \tau_{i-1},\ i \geq 1.$$

In this case, we say that $\bar{\gamma}$ is a run of M over $(\bar{x}, \bar{\tau})$ from γ_0.

Note that, in a timed sequence $(\bar{x}, \bar{\tau})$, time evolves by $(\tau_i - \tau_{i-1})$ units from the moment when x_{i-1} occurred until x_i occurs (for $i > 1$). Intuitively a run captures the system evolution, as follows:

1. it starts at state s_0, with all clocks set to zero;
2. time evolves by $\tau_1 - \tau_0 = \tau_1$ units;
3. at instant τ_1 the system changes to state s_1 on input x_1 while resetting clocks in K_1 to zero;
4. time evolves by another $\tau_2 - \tau_1$ units;
5. at instant $\tau_1 + (\tau_2 - \tau_1) = \tau_2$ the system changes to state s_2 on input x_2 while resetting clocks in K_2 to zero;
6. and so on.

We can see that:

- a change of state can only occur when the transition (s, x, K, δ, s') is enabled, i.e., when δ is satisfied in the present configuration;
- clocks can be reset to zero in any transition;
- any clock reading is the elapsed time since the last instant it was reset to zero; and
- all clocks increase uniformly according to a global time frame.

3 The Timed EFSM model

In the previous sections, we have presented two formalisms: EFSMs and ETTSs. While EFSMs capture the relationships between inputs, outputs and context variables, ETTSs offer a treatment of time evolution and its constraints. We observe that there are several methods and techniques for deriving tests from (E)FSM models (e.g., [27, 17, 8, 26]). However, the derivation of test cases from (E)TTSs is less established, although some works have considered it (e.g., [30, 7, 18]). It is worth combining both ETTSs and EFSMs formalisms in order to benefit from the power of both models in terms of expressiveness. This section redefines the EFSM model in order to capture real-time. We use the ETTS definition as inspiration for this purpose.

3.1 Creating a TEFSM model from an ETTS and an EFSM model

Let X be a finite set of inputs, Y be a finite set of outputs, C be a finite set of clocks, R be a finite set of parameters, and V be a finite set of context variables. A Timed Extended Finite State Machine, or TEFSM, M over X, Y, R, V, C, and the associated valuation domains is a tuple $(S, T, Inv, s_0, \nu_0, \lambda_0)$, where S and T are finite sets of states and transitions, respectively, Inv is a finite set of invariant conditions associated with states and, $s_0 \in S$ is the initial state, $\nu_0 = [C \mapsto 0]$ is the initial clock interpretation and λ_0 is an initial context variable valuation. In the TEFSM model: (i) the dynamic behavior is given by clocks and their resetting, as in the ETTS model; and (ii) the data and control flow are given by parameters and context variables, as in the EFSM model. A transition $t \in T$ is expressed by a tuple $(s, x, Q, K, op, y, up, s')$, where:

- s, x, s' and K are as defined in the ETTS formalism; see Section 2.2;
- op, y, and up are as defined in the EFSM formalism; see Section 2.1;
- $Q : D_x \times [C \mapsto \mathbb{Q}] \times D_V \rightarrow \{True, False\}$ is the predicate of the transition.

It can be seen that the TEFSM model comprises the EFSM formalism. That is, given a EFSM M over X, Y, R, V and some valuation domains, as defined in Section 2.1, we can construct a TEFSM model $\hat{M}$ over the same sets X, Y, R, V and the corresponding domains, by letting the clock set C be simply $\{c\}$. For each transition $t = (s, x, P, op, y, up, s')$ in M, we define a transition $\hat{t} = (s, x, Q, K, op, y, up, s')$ in $\hat{M}$ by letting $Q(p_x, \nu, \lambda) = P(p_x, \lambda)$, for any (p_x, ν, λ) in $D_x \times [C \mapsto \mathbb{Q}] \times D_V$. We also let $K = \emptyset$. Clearly, for any $p_x \in D_x$, $\lambda_v \in D_V$ and any clock interpretation $\nu \in [C \mapsto \mathbb{Q}]$, we have that $Q(p_x, \nu, \lambda_v)$ is true iff $P(p_x, \lambda_v)$ is true. For each state $s \in S$ in M, we define the invariant condition $\hat{Inv}(s) = (c \geq 0)$ in $\hat{M}$. Clearly, $\nu \vDash \hat{Inv}(s)$ for any $\nu \in [C \mapsto \mathbb{Q}]$ and $s \in S$.

Also, any ETTS model can be cast as a TEFSM model. For that, let $M = (S, s_0, X, C, Inv, \longrightarrow)$ be an ETTS model. Take a trivial common domain $\{0\}$ for all parameters and context

variables, a single output symbol $Y = \{o\}$ and a single context variable $V = \{v\}$. For each parameter $z \in X \cup Y$, we define $R_z = \{z\}$. Then, the set of z-valuations is the singleton $D_z = \{p_z\}$, where p_z maps z to 0, for all $z \in X \cup \{o\}$. Similarly, $D_V = \{\lambda_v\}$, where λ_v maps v to 0. Now, a transition $t = (s, x, K, \delta, s')$ in M gives rise to a corresponding transition $\hat{t} = (s, x, Q, K, op, o, up, s')$ in $\hat{M}$, where:

- op maps (p_x, λ_v) to p_o;
- up maps (p_x, λ_v) to λ_v; and
- Q maps (p_x, ν, λ_v) to $True$ iff $\nu \vDash \delta$.

Here, the set of invariant conditions Inv for M is the same for $\hat{M}$. The initial state s_0 in $\hat{M}$ is the same initial state s_0 from M.

A configuration of a TEFSM M is a triple (s, ν, λ), where s is a state, ν is a clock interpretation and λ is a context variable valuation. The initial configuration is (s_0, ν_0, λ_0), where s_0 is the initial state of M, ν_0 is the initial clock interpretation of M and λ_0 is an initial context variable valuation of M. A configuration (s, ν, λ) is valid iff $\nu \vDash Inv(s)$. Let $\Gamma \subseteq S \times [C \mapsto \mathbb{Q}] \times D_V$ be the set of configurations of M.

3.2 The Operational Semantics for TEFSM models

Considering the dynamic behavior of ETTS models and the data and control flow of EFSM models, we define the operational semantics of a TEFSM M as follows.

Definition 1 *Let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2$, be two configurations of M. There is an implicit move from γ_1 to γ_2 iff*

1. $s_1 = s_2$,
2. $\lambda_1 = \lambda_2$,
3. $\nu_2 = \nu_1 + \tau$, *for some* $\tau \in \mathbb{Q}^{>0}$, *and*
4. $\nu_2 + \eta \vDash Inv(s_1)$, *for all* η, $0 \leq \eta \leq \tau$.

We denote such an implicit move by $\gamma_1 \xrightarrow[\tau]{} \gamma_2$.

Definition 2 *Let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2$, be two configurations of M. Let (x, p_x) be a parameterized input and (y, p_y) be a parameterized output. There is an explicit move from γ_1 to γ_2 over (x, p_x) and yielding (y, p_y) iff there is a transition $(s_1, x, Q, K, op, y, up, s_2)$ in T such that:*

1. $\nu_2 = [K \mapsto 0]\nu_1$,
2. $\nu_2 \vDash Inv(s_2)$,
3. Q *maps* (p_x, ν_1, λ_1) *to* $True$,
4. op *maps* (p_x, λ_1) *to* p_y, *and*
5. up *maps* (p_x, λ_1) *to* λ_2.

We denote such an explicit move by $\gamma_1 \xrightarrow{\chi/\xi} \gamma_2$, *where* $\chi = (x, p_x)$ *and* $\xi = (y, p_y)$.

Definition 3 *Let* $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma$, $i = 1, 2, 3$; $\tau \in \mathbb{Q}^{>0}$, (x, p_x) *a parameterized input and* (y, p_y) *a parameterized output. If* $\gamma_1 \underset{\tau}{\longrightarrow} \gamma_2$ *and* $\gamma_2 \xrightarrow{\chi/\xi} \gamma_3$, *where* $\chi = (x, p_x)$ *and* $\xi = (y, p_y)$, *then we say that there is a move from* γ_1 *to* γ_3 *and indicate this by* $\gamma_1 \xrightarrow[\tau]{\chi/\xi} \gamma_3$.

Some of the decorations over and under $\longrightarrow$ may be dropped if they are clear from the context.

A parameterized input sequence is any sequence $\bar{\rho} = \rho_1\rho_2 \ldots$ where each ρ_i is a parameterized input. A parameterized timed input sequence, or timed input, is a pair $(\bar{\rho}, \bar{\tau})$ where $\bar{\rho}$ is a parameterized input and $\bar{\tau}$ is a time sequence. Similar definitions hold for parameterized outputs. In particular a timed output is a parameterized timed output sequence.

A sequence of configurations $\bar{\gamma} = \gamma_0\gamma_1\gamma_2 \ldots$ is a run of M iff there are a timed input $(\bar{\rho}, \bar{\tau})$ and a parameterized output sequence $\bar{\mu}$ such that

$$\gamma_{i-1} \xrightarrow[\theta_i]{\rho_i/\mu_i} \gamma_i, \text{ where } \theta_i = \tau_i - \tau_{i-1}, \text{ for all } i \geq 1.$$

We say that the run is over the timed input $(\bar{\rho}, \bar{\tau})$ and produces the timed output $(\bar{\mu}, \bar{\tau})$. We also say that $(\bar{\mu}, \bar{\tau})$, or $\bar{\mu}$, is produced by M from γ_0 in response to $(\bar{\rho}, \bar{\tau})$.

Some notions from the EFSM and ETTS models are extended to the TEFSM model:

- A TEFSM M is said to be predicate complete if, from any configuration (s, ν, λ) and given any parameterized input (x, p), there is a delay τ and a transition $(s, x, Q, K, op, y, up, s')$ such that Q evaluates $(p, \nu + \tau, \lambda)$ to $True$ and $\nu + \eta \vDash Inv(s)$, for all $0 \leq \eta \leq \tau$.
- The TEFSM M is complete if, for each state s there is a transition leaving s on any input symbol x.
- We say M is deterministic if, for any configuration (s, ν, λ), any parameterized input (x, p), and any time instant τ, there are no two different transitions $(s, x, Q_1, K_1, op_1, y_1, up_1, s_1)$ and $(s, x, Q_2, K_2, op_2, y_2, up_2, s_2)$ such that both Q_1 and Q_2 evaluate $(p, \nu + \tau, \lambda)$ to $True$.

- And, we say M is observable if, for any configuration (p, ν, λ), any parameterized input (x, p) there are no two transitions $(s, x, Q_1, K_1, op_1, y_1, up_1, s_1)$ and $(s, x, Q_2, K_2, op_2, y_2, up_2, s_2)$ with $y_1 \neq y_2$ and with Q_1 and Q_2 both evaluating (p, ν, λ) to $True$.

3.3 Configuration Distinguishability in the TEFSM model

Distinguishability of configurations in the Timed Extended Finite State Machine model is defined over parameterized input sequences. Two configurations γ and γ' of two distinct machines M and M', respectively, are distinguishable over a timed input $(\bar{\rho}, \bar{\tau})$ if the corresponding timed outputs $(\bar{\mu}, \bar{\tau})$ and $(\bar{\mu}', \bar{\tau}')$, produced by M and M' over $(\bar{\rho}, \bar{\tau})$ from γ and γ', respectively, are not compatible, in a sense to be defined shortly. We also say that $(\bar{\rho}, \bar{\tau})$ is a timed input separating those two configurations. We formalize these notions in the sequel, extending the definitions in [27]. Given a context variable valuation λ and a set of variables U, the U-projection of λ is the valuation obtained from λ by retaining the variables that are in the set U, denoted by $\lambda \downarrow U$. Similarly, for input symbols and their valuations, and for output symbols and the corresponding valuations.

Definition 4 *Let y and y' be outputs of TEFSMs M and M', respectively. Let R and R' be the sets of parameters associated, respectively, with y and y'. The parameterized outputs (y, p) and (y', p') are said to be compatible if $y = y'$ and $p \downarrow R' = p' \downarrow R$. Two parameterized output sequences, $(y_1, p_1) \ldots (y_k, p_k)$ of M and $(y'_1, p'_1) \ldots (y'_k, p'_k)$ of M' are compatible if, for all $i = 1, \ldots, k$, the parameterized outputs (y_i, p_i) and (y'_i, p'_i) are compatible.*

Intuitively, parameterized outputs are compatible when the output symbol is the same, and the output valuation agrees on all common output symbols. Distinguishability of configurations is defined as follows.

Definition 5 *Given a timed input $\bar{\alpha} = (\bar{\rho}, \bar{\tau})$, a configuration γ of M and a configuration γ' of M' are distinguishable by $\bar{\alpha}$ if parameterized output sequence produced by M from γ in response to $\bar{\alpha}$ is not compatible with any parameterized output sequence that can be produced by M' from γ' in response to $\bar{\alpha}$. The timed input $\bar{\alpha}$ is said to be a sequence separating γ from γ'.*

4 Timed Extended FSM Product

In Section 5 we extend to TEFSMs the method for the derivation of configuration confirming sequences defined in [27]. Since this method requires the notion of product machines, in this section we present the necessary extension of that notion to TEFSMs.

In the product of TEFSMs, the occurrence of implicit transitions can be ignored, since the global time frame which is used for all clock variables is the same for both TEFSMs. This guarantees that the system evolution is maintained during implicit transitions.

Let $M^i = (S^i, Inv^i, T^i)$, $i = 1, 2$, and $\gamma^i = (s_0^i, \nu_0^i, \lambda_0^i)$, $i = 1, 2$, be two TEFSMs and their corresponding initial configurations. The product machine is denoted by $M^1 \times M^2$. We will use superscript 1 to denote elements of M^1, like R^1 is the set of parameters for M^1. Likewise, superscript 2 will indicate objects associated with M^2, like V^2 is the set of context variables of M^2. The superscript $1, 2$ is reserved for the product machine $M^1 \times M^2$.

The set of input symbols of $M^{1,2}$ is $X^{1,2} = X^1 \cup X^2$. Likewise, $Y^{1,2} = Y^1 \cup Y^2$. The set of parameters of $M^{1,2}$ is given by $R^{1,2} = R^1 \cup R^2$, with the proviso that for all $z \in R^1 \cap R^2$, the valuations of z in M^1 and M^2 have a common domain. It is clear that we are using the same parameter domains in $M^{1,2}$ as they were in M^1 and M^2. For any $z \in X^{1,2} \cup Y^{1,2}$, we let $R_z^{1,2} = R_z^1 \cup R_z^2$. Note that, given a valuation $r_z^{1,2}$ for elements in $R_z^{1,2}$ we can get valuations $r_z^1 = r_z^{1,2} \downarrow R_z^1$ and $r_z^2 = r_z^{1,2} \downarrow R_z^2$, for machines M^1 and M^2, respectively, and, moreover, $r_z^{1,2} = r_z^1 \cup r_z^2$. Similarly for clock interpretations and context variable valuations. We assume that clocks and context variables are disjoint, i.e., $C^{1,2} = C^1 \cup C^2$, with $C^1 \cap C^2 = \emptyset$, and $V^{1,2} = V^1 \cup V^2$, with $V^1 \cap V^2 = \emptyset$. As for the valuation domains, they are the same as in M^1 as in M^2. The set of states of $M^{1,2}$ is given by $S^{1,2} = S^1 \times (S^2 \cup \{fail\})$, where *fail* is a new state. The set of invariant conditions $Inv^{1,2}$ of $M^{1,2}$ maps $S^{1,2}$ to $\Phi(C^{1,2})$, and it is given by $Inv^{1,2}(s_1, s_2) = Inv^1(s_1) \wedge Inv^2(s_2)$, for all $(s_1, s_2) \in S^{1,2}$. Moreover, $Inv^{1,2}(s_1, fail) = Inv^1(s_1)$, for all $s_1 \in S^1$.

The initial configuration of $M^{1,2}$ will be given by $\gamma_0^{1,2} = ((s_0^1, s_0^2), (\nu_0^{1,2}, \lambda_0^{1,2}))$, where $\nu_0^{1,2} = \nu_0^1 \cup \nu_0^2$ and $\lambda_0^{1,2} = \lambda_0^1 \cup \lambda_0^2$. Note that we can take unions here, since clock and context variables are disjoint in M^1 and M^2.

It remains to specify the transitions of $M^{1,2}$. Let $(s_1^i, x, Q^i, K^i, op^i, y^i, up^i, s_2^i)$, $i = 1, 2$, be transitions of M^1 and M^2, both with the same input x. In the following definition we will be considering a parameterized input $(x, p_x^{1,2})$, a clock interpretation $\nu^{1,2}$ and a context variable valuation $\lambda^{1,2}$, all for the machine $M^{1,2}$. We also let $p_x^1 = p_x^{1,2} \downarrow R_x^1$ and $p_x^2 = p_x^{1,2} \downarrow R_x^2$. Likewise, we let $\nu^1 = \nu^{1,2} \downarrow C^1$ and $\nu^2 = \nu^{1,2} \downarrow C^2$, and also $\lambda^1 = \lambda^{1,2} \downarrow V^1$ and $\lambda^2 = \lambda^{1,2} \downarrow V^2$. There are two cases:

case 1: $y^1 = y^2$ and $op^1(p, \lambda) \downarrow R_{1,2} = op^2(p, \lambda) \downarrow R_{1,2}$, for all $(p, \lambda) \in D_x \times D_V$ where $R_{1,2} = R_{y^1}^1 \cap R_{y^2}^2$. That is, the output symbol is the same and the output valuations of both transitions are the same on each common output parameter. We add two transitions to $T^{1,2}$,

1. $((s_1^1, s_1^2), x, Q, K, op, y^1, up, (s_2^1, s_2^2))$, where:

   (a) $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1) \wedge Q^2(p_x^2, \nu^2, \lambda^2)$

(b) $K = K^1 \cup K^2$

(c) $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1) \cup op^2(p_x^2, \lambda^2)$. Recall that op^1 and op^2 coincide on common output parameters and so we can safely take the union.

(d) $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1) \cup up^2(p_x^2, \lambda^2)$. Recall that $V^1 \cap V^2 = \emptyset$.

2. $((s_1^1, s_1^2), x, Q, K, op, y^1, up, (s_2^1, fail))$, where:

(a) $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1) \wedge (\neg Q^2(p_x^2, \nu^2, \lambda^2))$

(b) $K = K^1$

(c) $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1)$

(d) $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1)$

case 2: Else, when the output valuations or the output symbols do not match, we add the transition $((s_1^1, s_1^2), x, Q, K, op, y, up, (s_2^1, fail))$ to $T^{1,2}$, where:

1. $Q(p_x^{1,2}, \nu^{1,2}, \lambda^{1,2}) = Q^1(p_x^1, \nu^1, \lambda^1)$
2. $K = K^1$
3. $op(p_x^{1,2}, \lambda^{1,2}) = op^1(p_x^1, \lambda^1)$
4. $up(p_x^{1,2}, \lambda^{1,2}) = up^1(p_x^1, \lambda^1)$

Moreover, if $(s_1^1, x, Q, K, op, y, up, s_2^1)$ is a transition of M^1, we add to $M^{1,2}$ the transition $((s_1^1, fail), x, Q, K, op, y, up, (s_2^1, fail))$.

Suppose that the product machine is in the state (s_1^1, s_1^2), and on input $(x, p_x^{1,2})$ we find that M^1, on state s_1^1, has a transition on input (s_1^1, p_x^1), where p_x^1 is the reduction of $p_x^{1,2}$ to the parameters associated with x in M^1. Similarly, M^2, on state s_1^2, has a transition on (x, p_x^2). Moreover, the output of these transitions agree on the output symbol y, and also on valuations of any common output parameter of y in M^1 and in M^2. In this situation, we would want the product machine $M^{1,2}$ to enact both transitions of M^1 and M^2, componentwise. For that: (i) the same clocks are reset; (ii) the output parameter valuations are copied from M^1 and M^2; and (iii) both context updates are also carried over to $M^{1,2}$. But we can only enable this action in $M^{1,2}$ if both transitions in M^1 and M^2 are enabled. This is case 1(i).

Otherwise, we consider the situation where the transition in M^1 is enabled, but the one in M^2 is not. Here, we follow case 1(ii), and make the product machine $M^{1,2}$ enact the behavior of M^1 using for that the first state component, while the second component is marked as *fail*, thereby ignoring the transition from M^2. Note that, in this scenario, M^1 might have taken its transition, while M^2 would be forbidden to do so, even when their external behavior would have

been indistinguishable. After the second state component is set to *fail*, $M^{1,2}$ behaves essentially as M^1.

Finally, when the product machine is in state (s_1^1, s_1^2), and we are considering an input $(x, p_x^{1,2})$, and we have picked two transitions from M^1 and M^2, starting respectively at s_1^1 and s_1^2, and whose output symbols or output parameter valuations do not match as above, then we proceed as in case 2. This is similar to case 1(ii) in that the second state component in $M^{1,2}$ is marked as *fail*, and $M^{1,2}$ uses the first state component to behave as M^1, from this moment on.

Consider configurations $\gamma^i = (s^i, \nu^i, \lambda^i)$ of machine M^i, $i = 1, 2$. Let $\bar{\rho} = (\bar{x}, \bar{p_x})$ be a parameterized input sequence for $M^1 \times M^2$, and let $\bar{\alpha} = (\bar{\rho}, \bar{\tau})$ is a timed input for $M^1 \times M^2$. Note that, M^1 and M^2 can be the same machine with different initial configurations. We say that $\bar{\alpha}$ is a separating sequence for γ^1 and γ^2 iff there is a run $\bar{\gamma} = \gamma_0\gamma_1 \ldots$ of $M^{1,2}$ over $\bar{\alpha}$, where $\gamma_0 = ((s^1, s^2), \nu^1 \cup \nu^2, \lambda^1 \cup \lambda^2)$ and for some $i \geq 1$, γ_i is a configuration of $M^{1,2}$ whose state is $(s_j^1, fail)$ for some $s_j^1 \in S^1$.

The problem of determining a separating sequence for two configurations of a given TEFSM M can be reduced to a reachability problem. The reachability analysis is tractable but hard for EFSMs [23]. Indeed, for TEFSMs it is intractable. This is due to the temporal aspect within the new model. Another difficulty is the combinatorial explosion in the number of states in product machines. Some approaches try to overcome this difficulty by relaxing their restrictions. Approximation algorithms are also used when doing reachability analysis. Other approaches adapted known algorithms in order to manipulate symbolic data structures [34, 9, 35].

Other simpler contexts [25, 3] present algorithms to obtain separating sequences. We postulate that these ideas can be adapted and extended in order to obtain separating sequences in the TEFSM formalism. Such separating sequences would be the result of the test case generation procedure. Moreover, we have been working with the notion of automata discretization in order to overcome the problem of infinite time instants. In addition, it is possible to modify conventional algorithms to reduce the state space generated by the product machine. Another alternative to obtain tractability in a timed approach for finding separating sequences is through the use of suspicious configurations [5]. In this case, we can choose a set of suspicious states, representing an important class fault, based on the expertise of test designers and on assumptions of implementations faults, as seen in [13, 31].

5 Test Generation

This section outlines the main concepts for test case generation. First, we present some discussion on the main rationale of conformance testing. Second, we discuss the notion of confirming configurations, and how it is applied. At last, we discuss deriving test sequences by model-checking for TEFSMs.

5.1 Conformance Testing

Conformance testing aims at determining whether an implementation behaves in accordance with a given specification [21, 15]. In general, an implementation is regarded as a black box, of which only input/output interfaces are known. In this situation, to verify whether an implementation is in conformance to a specification usually requires an infinite set of test cases in order to exhaust all error possibilities in the implementation. To overcome this problem, one possibility is to define a set of test hypotheses in order to reduce the number of test cases to be considered [13]. Test hypotheses strike a balance between two conflicting aspects. On the one hand, test hypotheses must be defined to be restrictive enough to render the method feasible and tractable. On the other hand, these hypotheses must be as less restrictive as possible, in such a way to be applicable to the largest possible set of implementations.

Conformance testing is guided by a conformance relation between the implementation and the specification [13]. In order to decide whether an implementation is in conformance to a specification, we observe the implementation's outputs to some applied inputs. Considering real-time systems, it must be also verified whether an implementation when stimulated by inputs responds with the expected outputs within an allowed time interval.

The problem of using a conformance relation is the number of test sequences which should be obtained in order to verify whether each possible implementation is in conformance to a given specification. This problem is worse for timed systems, where there are infinite time instants for a transition to occur. To overcome this problem, we also need to enforce certain hypotheses about the implementation, as discussed in Section 5. This set of hypotheses will reduce the number of possible faults to be considered over the implementation and will render the method feasible in practical cases.

Several methods employ identification sequences to generate test cases from models. An identification sequence has the property of determining the correctness of the configuration reached after some input sequence is taken. Identification sequences may be defined as characterization sets [8, 13], as distinguishing sequences [17] or as confirming configuration sequences (CCSs) [27], depending on the model and the generation method. A CCS that is investigated in this paper is a sequence that can increase the confidence that the correct configuration has been reached in the implementation.

5.2 Configuration Confirming Sequences

A configuration confirming sequence (CCS) is a timed input that can be applied to the implementation in order to increase the confidence on its correctness. A CCS can be derived from the product of two machines, one being a specification and the other an undesirable configuration. However, unlike the FSM models where a finite set of undesirable configurations can be postulated, with EFSM models and TEFSM models it is not possible, or desirable, to determine all

undesirable configurations. To overcome this problem, a finite set of suspicious configurations is considered [27]. A set of suspicious configurations is derived from the specification to model suspicious implementations which can potentially have faults, reflecting the test designer's assumptions about the implementation faults. The suspicious configurations are extracted from the specification using a set of test hypotheses based on the fault model (e.g., [13]) and relying on the test designer's expertise.

These hypotheses define equivalence classes of implementations that must be put under testing, and they are used to reduce the number of possible implementations that need to be considered. In this work, we assume the following test hypotheses:

1. Specifications and suspicious implementations are modeled by TEFSMs;

2. The number of clocks in the specification must be less than or equal to the number of clocks in the suspicious implementations; and

3. The same alphabets are used in both specification and suspicious implementations.

Given a configuration and a suspicious configuration, deriving a CCS can be reduced to the problem of finding a path in the product of two distinct TEFSMs, or of the same core TEFSM with distinct initial configurations. Such a sought path would run from the initial state to a fail state. If the fail state can not be reached, then the suspicious configuration is equivalent to the original configuration. However, if a fail state is reachable, the model-checking algorithm will produce a counter-example, as a sequence of transitions that leads to this fail state [10]. This sequence would make a test case for the suspicious configuration. However, it is still necessary to identify in which moment each transition was taken, as well as the valuation of the input parameters associated with each input symbol. Gathering of this information forms a set of test cases. The test case is then used to exercise a real implementation, and the outputs are compared with the outputs produced by the specification over the same data. If a disagreement is found between corresponding outputs, then a fault has been identified.

5.3 Model-checking

Design errors frequently occur when conventional simulation and testing techniques are used to check safety systems. Model checking is a set of techniques for the automatic analysis of reactive and real-time systems. Some model checking algorithms have been modified to discover such errors, thus providing more quality and accuracy in system verifications. In general, given a transition system and a property, the model checking problem is to decide whether the property holds or not in the model represented by the transition system. If not, the model checking algorithm provides a counterexample, i.e. an execution over the model that violates the property [23].

Reachability analysis is a special kind of model-checking method that can be applied in a formal model. In general, given a special state to be found in a model, the reachability analysis decides if it is possible to move from the initial state to the final special state.

To summarize, to automatically test implementations based on a specification represented by a machine M, the following steps are performed:

1. An empty set TC of test cases is defined.
2. Given a configuration γ of M, a set of suspicious configurations Γ is defined, based on test hypotheses, fault models and some specific test engineer's objectives.
3. For each suspicious configuration $\gamma_s \in \Gamma$, the product of M with itself is constructed, having γ as the initial configuration of the first instance of M in the product, and having γ_s as the initial configuration of the second instance.
4. Reachability analysis is carried out, in order to find a path to a fail state in the product machine. If such a path is found, it is added to TC.
5. For each $tc \in TC$, a time and an input parameter valuation sequences are derived so as to satisfy the predicates along the path specified by tc.
6. Each path in TC, with its associated data, is applied to the real implementation under testing.

6 An Example

We are given two TEFSMs M and N, where M is a specification and N is a suspicious implementation of M. We obtain the product of these machines, $M \times N$, by applying our method. In this example, as is usual in practice, N has the same transitions as M. They differ only in their associated initial configurations. Accordingly, we will denote the product by $M^0 \times M^1$, where M^0 is the specification and M^1 is the suspicious configuration. The TEFSM M depicted in Figure 1.

It has three states and seven transitions. The input set is $\{a, b\}$ and the output set is $\{c, d, e\}$. Furthermore, M has two clock variables, x and y, and one context variable w. There are no parameters associated with the input symbols, i.e. $R_a = R_b = \emptyset$. Likewise, $R_c = R_d = \emptyset$. For each state s in M, the control remains in s whenever its invariant condition is satisfied. The output e has only one associated parameter. In this case, it is not necessary to name the parameter. Instead, in Figure 1 and in the sequel we write $e(w)$ to indicate that the current value of the context variable w is to be assigned to the parameter associated with e. In the figure, each arrow is labeled by a sequence of items. The first three are always the input symbol, the

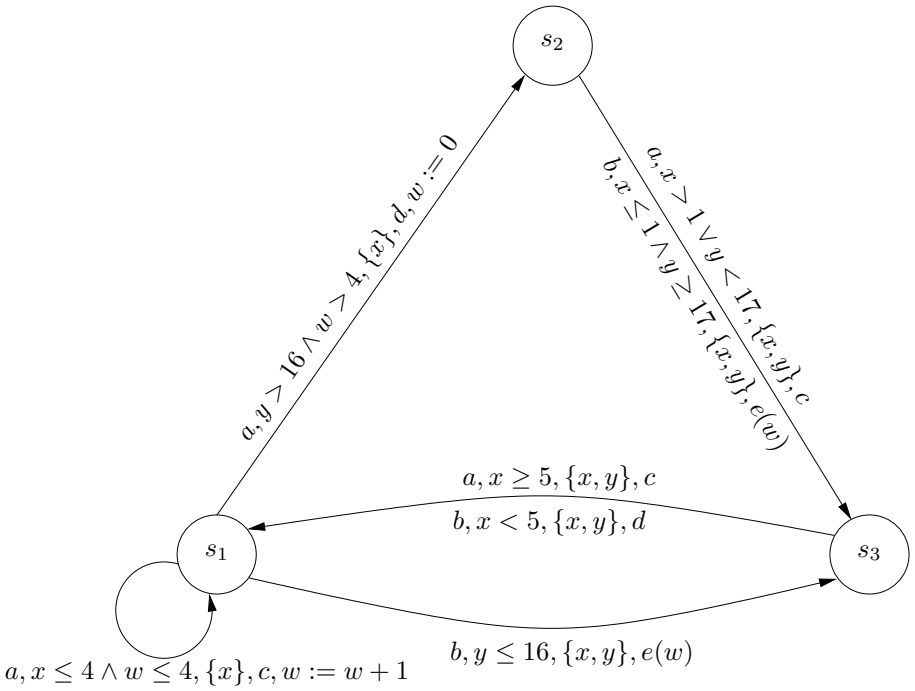


Figure 1: The TEFSM M.

predicate function and the set of clocks to be reset in the transition, respectively. Next, comes the output symbols, either c or d, and we write directly $e(w)$ to indicate both the output symbol and the value of its parameter. Finally, if the value of the context variable w is altered by the transition, this is indicated by the attribution that appears at the end of the label; if the value of w is not altered by the transition we simply omit the trivial expression $w := w$.

A configuration of M is given by a state, a clock interpretation and a context variable valuation. Hence, a configuration of M will be denoted by $(s, (n, m), k)$ indicating that the machine is in state s, n and m are the values for the clock variables x and y, respectively, and k is the value for the context variable w. The integers are selected as a common valuation domain. In the configuration $(s_1, (3, 2), 4)$ the transition $a, x \leq 4 \wedge w \leq 4, \{x\}, c, w := w + 1$, from s_1 to itself, is enabled. Likewise, the transition $b, y \leq 16, \{x, y\}, e(w)$, from s_1 to s_3, is also enabled.

For the product, let M^0 designate M with the initial configuration $(s_1, (0, 0), 2)$, and let M^1 designate M with initial configuration $(s_1, (4, 2), 5)$. The TEFSM product of $M^0 \times M^1$ is shown in Figure 2. To simplify the notation in the example, we will use subscript i to denote items of machine M^i, for $i = 0, 1$, e.g. x_1 represents the clock variable x of M_1, while w_0 denotes the variable w in M^0.

The initial configuration of $M^0 \times M^1$ is denoted by $((s_1, s_1), (0, 0, 2, 4), (2, 5))$, where we

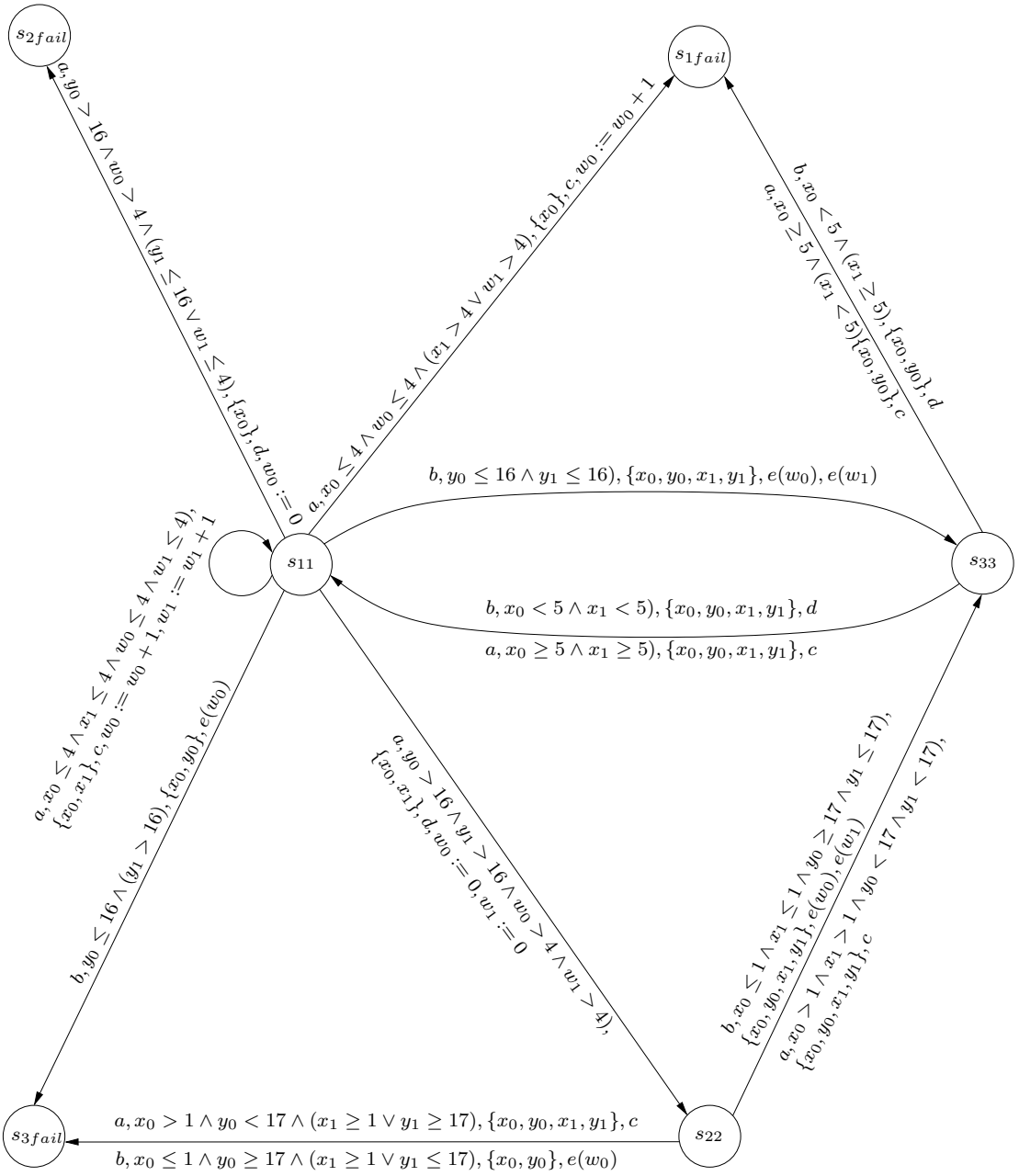


Figure 2: The TEFSM product of M with itself.

list first the items corresponding to M^0, followed by the items associated with M^1. Note that, in the figure, states are represented by subscripts, e.g., the state (s_1, s_1) in the product is named s_{11}.

By inspection of the product, we can see that the input b enables the transition to fire, since clock conditions on y_0 and y_1 are satisfied for the initial configuration. After that the transition is taken, the new configuration is given by $((s_3, s_3), (0, 0, 2, 0), (0, 5))$. It is easy to see that neither input a nor input b will enable transitions to fire so as to reach, directly, a fail state. Note that, every clock variable was reset to zero, and transition guards are excluding for clock variables x_0 and x_1. If the input b occurs within less than 5 time units, the configuration becomes $((s_1, s_1), (0, 0, 2, 0), (0, 5))$. Otherwise, if the time evolves for more than 5 time units, only the input a could stimulate the machine to change configurations. The new configuration would still be $((s_1, s_1), (0, 0, 2, 0), (0, 5))$. Both transitions would drive the control back to the initial state, where the transition stimulated by input b is the unique one enabled to fire. This cycle would be executed repeatedly and a fail state would not be reached.

Another possibility is to take the transition on the input a. It is easy to see that the input a separates the configurations $(0, 0, 2)$ from $(4, 2, 5)$. The final configuration reached is $((s_1, fail), (0, 0, 3, 4), (2, 5))$. On the other hand, the control can be kept within the state (s_1, s_1), by a continuous time evolution. After that, the stimulation by input a enables the transition to fire, and the configuration $((s_1, s_1), (1, 1, 2, 5), (3, 5))$ can be reached. Then, the transition from state (s_1, s_1), on input a and with associated predicate $x_0 \leq 4 \wedge w_0 \leq 4 \wedge (w_1 > 4 \vee x_1 > 4)$ is enabled and takes the machine to the fail state $(s_1, fail)$. Here, only the clock variable x_0 is reset, and the context variable w_0 is updated by one unit. The new configuration will be $((s_1, fail), (0, 1, 3, 5), (3, 5))$. From here, we see that input a separates the configuration $(0, 0, 2)$ from $(4, 2, 5)$, after some time passes. The new configuration that can be reached in this case is $((s_1, fail), (0, 1, 3, 5), (3, 5))$.

If we consider another situation, where the initial configurations of M^0 and M^1, respectively, are given by $(0, 0, 2)$ and $(4, 2, 4)$, another run of $M^0 \times M^1$ will also reach the fail state. In this case, a reachability analysis shows that the fail state of $M^0 \times M^1$ can only be reached when a sequence of one or two consecutive inputs a is applied.

In the example, M^0 represents the specification, M^1 represents a suspicious implementation, and the product $M^0 \times M^1$ is used to find sequences of configurations that show non conformance between a suspicious implementation and the specification. We can derive traces from the reachability analysis of $M^0 \times M^1$. The resulting traces are runs that reach the fail state in the product machine, starting from the initial configurations of the participating TEFSMs.

7 Concluding Remarks

The ability to derive test cases from formal models opens the possibility that we can construct more rigorous and dependable systems, by providing a sound basis for the validation of the systems' behaviors. There is a direct relationship between the kinds of systems that a given model can deal with and the availability of methods for deriving test cases. The FSM and EFSM models are well-established and have been intensively investigated. One important feature they both lack is the ability to deal with time. In this paper we define TEFSMs as a model that extends the EFSM model with the notion of time. From that, we discussed an extended method for deriving configuration confirming sequences for TEFSMs, a step toward automating the generation of test cases from these models.

Although we can argue that both the model and the generation method can be used, we do not have answers for pragmatic questions, such as (i) how difficult is it to describe a system using TEFSMs and (ii) how large are the models we can handle. To answer these questions, it is necessary to deepen the investigations and implement adequate supporting software tools. We are currently working in this direction.

Other aspects that can be investigated include how to allow time constraint to be defined over outputs. We note that our definition does not deal with constraints that may reflect output response that is not instantaneous. The input and output occur in the same time instant. We are considering how this extension might impact the test case generation methods.

References

[1] Alur, R., *Timed automata*, in: *CAV*, number 1633 in LNCS, 1999, pp. 8–22.

[2] Alur, R. and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), pp. 183–235.
URL `citeseer.ist.psu.edu/alur94theory.html`

[3] Alur, R., M. McDougall and Z. Yang, *Exploiting behavioral hierarchy for efficient model checking.*, in: *CAV*, 2002, pp. 338–342.

[4] Behrmann, G., K. G. Larsen, J. Pearson, C. Weise and W. Yi, *Efficient timed reachability analysis using clock difference diagrams*, in: *Computer Aided Verification*, 1999, pp. 341–353.
URL `citeseer.ist.psu.edu/article/behrmann99efficient.html`

[5] Bonifácio, A. L., A. V. Moura, A. d. S. Simão and J. C. Maldonado, *Conformance Testing by Model Checking Timed Extended Finite State Machines*, in: *Brazilian Symposium on Formal Methods (SBMF'06)*, Natal, 2006, pp. 43–58.

[6] Campos, S. V., M. Minea, W. Marrero, E. M. Clarke and H. Hiraishi, *Computing quantitative characteristics of finite-state real-time systems*, in: *Proc. 15th IEEE Real-Time Systems Symp.* (1994), pp. 266–270, san Juan, Porto Rico.

[7] Cardell-Oliver, R., *Conformance tests for real-time systems with timed automata specifications*, Formal Aspects of Computing **12** (2000), pp. 350–371.
URL `citeseer.ist.psu.edu/385816.html`

[8] Chow, T. S., *Testing software design modeled by finite-state machines*, IEEE Transactions on Software Engineering **4** (1978), pp. 178–187.

[9] Cimatti, A., E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella, *Integrating bdd-based and sat-based symbolic model checking.*, in: *FroCos*, 2002, pp. 49–56.

[10] da Silva, D. A. and P. D. L. Machado, *Towards test purpose generation from ctl properties for reactive systems.*, Electr. Notes Theor. Comput. Sci. **164** (2006), pp. 29–40.

[11] Dickhöfer, M. and T. Wilke, *Timed alternating tree automata: the automata-theoretic solution to the tctl model checking problem*, in: *26th ICALP*, LNCS **1644**, 1999, pp. 281–290.
URL `citeseer.ist.psu.edu/article/dickhfer99timed.html`

[12] Dorofeeva, R., K. El-Fakih and N. Yevtushenko, *An improved conformance testing method*, in: *Formal Techniques for Networked and Distributed Systems*, Lecture Notes in Computer Science **3731** (2005), pp. 204–218.

[13] En-Nouaary, A., R. Dssouli and F. Khendek, *Timed wp-method: Testing real-time systems*, IEEE Trans. Softw. Eng. **28** (2002), pp. 1023–1038.

[14] Fujiwara, S., G. V. Bochmann, F. Khendek, M. Amalou and A. Ghedamsi, *Test selection based on finite state models*, IEEE Transaction on Software Engineering **17** (1991).

[15] Gargantini, A., *Conformance testing*, in: M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, Lecture Notes in Computer Science **3472** (2005), pp. 87–111.

[16] Gill, A., "Introduction to the theory of finite-state machines," McGraw-Hill, New York, 1962.

[17] Gonnenc, G., *A method for the design of fault detection experiments*, IEEE Transactions on Computing **19** (1970), pp. 551–558.

[18] Higashino, T., A. Nakata, K. Taniguchi and A. R. Cavalli, *Generating test cases for a timed i/o automaton model*, in: *Proceedings of the IFIP TC6 12th International Workshop on Testing Communicating Systems* (1999), pp. 197–214.

[19] Hirai, T., *An application of temporal linear logic to Timed Petri Nets*, in: *Proceedings of the Petri Nets'99 Workshop on Applications of Petri Nets to Intelligent System Development*, 1999, pp. 2–13.

[20] Krichen, M., *State identification*, in: M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker and A. Pretschner, editors, *Model-Based Testing of Reactive Systems: Advanced Lectures*, Lecture Notes in Computer Science **3472** (2005), pp. 87–111.

[21] Krichen, M. and S. Tripakis, *Black-box conformance testing for real-time systems*, in: *Model Checking Software: 11th International SPIN Workshop*, number 2989 in Lecture Notes in Computer Science, Barcelona, Spain, 2004, pp. 109–126.

[22] McMillan, K. L., "Symbolic Model Checking: An Approach to the State Explosion Problem," Kluwer Academic, 1993.

[23] Merz, S., *Model checking: A tutorial overview*, in: F. C. et al., editor, *Modeling and Verification of Parallel Processes*, Lecture Notes in Computer Science **2067**, Springer-Verlag, Berlin, 2001 pp. 3–38.

[24] Møller, J. B., *Simplifying fixpoint computations in verification of real-time systems* (2002).
URL `http://citeseer.ist.psu.edu/540135.html`

[25] Nr, B., M. Dickhofer and T. Wilke, *The automata-theoretic method works for TCTL model checking* (1998).
URL `http://citeseer.ist.psu.edu/44733.html`

[26] Offutt, A. J., Y. Xiong and S. Liu, *Criteria for generating specification-based tests*, in: *Fifth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS '99)*, Las Vegas, NV, 1999, pp. 41–50.

[27] Petrenko, A., S. Boroday and R. Groz, *Confirming configurations in EFSM testing*, IEEE Trans. Softw. Eng. **30** (2004), pp. 29–42.

[28] Petrenko, A. and N. Yevtushenko, *Testing from partial deterministic fsm specifications*, IEEE Transactions on Computers **54** (2005).

[29] Tretmans, J., *Test generation with inputs, outputs, and quiescence.*, in: T. Margaria and B. Steffen, editors, *Tools and Algorithms for Construction and Analysis of Systems, Second*

International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings, Lecture Notes in Computer Science **1055** (1996), pp. 127–146.

[30] Tretmans, J., *Testing concurrent systems: A formal approach*, in: J. Baeten and S. Mauw, editors, *CONCUR'99 – 10^{th} Int. Conference on Concurrency Theory*, Lecture Notes in Computer Science **1664** (1999), pp. 46–65.

[31] Wang, C.-J. and M. T. Liu, *Generating test cases for efsm with given fault models.*, in: *INFOCOM*, 1993, pp. 774–781.

[32] Wang, F., *Efficient verification of timed automata with bdd-like data structures.*, STTT **6** (2004), pp. 77–97.

[33] Wang, F., *Formal verification of timed systems: A survey and perspective.*, Proceedings of the IEEE **92** (2004), pp. 1283–1307.

[34] Wang, F., *Symbolic parametric safety analysis of linear hybrid systems with bdd-like data-structures*, Software Engineering, IEEE Transactions on **31** (2005), pp. 38–51.

[35] Wang, F., *Under-approximation of the Greatest Fixpoints in Real-Time System Verification*, ArXiv Computer Science e-prints (2005), pp. 1060–+.

[36] Wang, F., G.-D. Hwang and F. Yu, *Tctl inevitability analysis of dense-time systems.*, in: *CIAA*, 2003, pp. 176–187.

Epílogo

Neste capítulo foi apresentada, em detalhes, uma abordagem de teste de conformidade para a geração de casos de teste baseado em confirmação de configurações. O objetivo foi a geração de testes para modelos de contexto e tempo. Neste estudo o modelo TEFSM proposto possui entradas e saídas associadas, e permite a especificação de sistemas temporizados, onde as restrições de tempo, em que ocorrem esses eventos, são importantes.

Os resultados alcançados compõem um primeiro passo na direção de se obter um modelo formal que capture a noção de tempo, juntamente com as transformações de contexto dado pelo modelo EFSM. Neste trabalho optamos pela técnica de *model-checking* para que caminhos no produto de TEFSM sejam encontrados, de tal forma que uma configuração suspeita resulte num contra-exemplo dado pela aplicação do algoritmo de model-checking.

No entanto, a proposta não aborda uma cobertura completa de falhas, e também não permite a ocorrência de eventos independentes, entre ações de entrada e saída. Por isso o interesse por mecanismos formais que possam fornecer uma melhor cobertura de falhas bem como flexibilizar a ocorrência autônoma de eventos de entrada e saída em sistemas reativos.

O próximo capítulo apresenta uma abordagem, também de discretização, porém, aplicada a modelos temporizados com transformações de contexto. Neste caso, o modelo, chamado TIOCA, é uma evolução dos TIOA, e a abordagem é diferente daquela apresentada para o modelo TEFSM, que evoluíram das EFSM.

Capítulo 5

Geração de Testes para Sistemas de Tempo e Contexto

Prólogo

Este trabalho foca a atividade de teste baseada em modelos, na busca por métodos eficientes de geração de casos de teste para sistemas críticos complexos. Neste caso a proposta utiliza um novo modelo, mais expressivo e complexo, chamado TIOCA, um autômato temporizado com variáveis de contexto, que também possui parâmetros associados às entradas e saídas, e permite além da evolução contínua de tempo, também as transformações de contexto. Nesta direção, não se conhece até então abordagens que lidem com aspectos de tempo e contexto ao mesmo tempo, como explorado neste trabalho.

A geração de casos de teste para tal modelo recai, novamente, no problema de explosão do número de estados, visto que agora, além do tempo contínuo, temos as variáveis de contexto, aumentando a complexidade dessa tarefa.

Para solucionar o problema da explosão de estados, propomos uma nova discretização baseada na discretização desenvolvida para TIOA. Ambas as propostas evitam a tradicional abordagem baseada nas regiões de relógio. Com isso, a relação de simulação entre o comportamento do modelo original e o seu correspondente autômato grid é estabelecido, o que nos dá a base para a geração dos testes. Mostramos então que modelos TIOCA podem simular de maneira homomórfica os seus respectivos grid, e vice-versa.

De posse do modelo discretizado, desenvolvemos uma técnica para a geração e aplicação dos casos de teste em implementações candidatas. Usamos também as propostas de teste para modelar propriedades específicas do sistema, bem como o conceito de produto para TIOCA. Assim, a extração dos testes ocorre sobre o produto discretizado que por sua vez são aplicados em implementações candidatas. Com a estratégia de realimentação das sequências, obtidas das implementações, nos autômatos grid, podemos verificar se as execuções encontradas são as

mesmas do modelo original. A relação entre o modelo original e o grid, dada pela simulação apresentada com provas detalhadas neste trabalho, garante a verificação de compatibilidade dos comportamentos dos modelos.

Este capítulo é composto pelo artigo, "Generating Test Suites for Timed Systems with Context Variables", que apresenta um novo método de discretização para o modelo TIOCA, permitindo a geração de casos de teste para sistemas de tempo e contexto. A idéia é utilizar um modelo discretizado e evitar a explosão combinatória de estados, gerando conjuntos de teste de forma eficiente. O artigo foi submetido para a *IEEE International Conference on Software Testing, Verification and Validation* (ICST'10), que será realizada em Paris, França, de 06 a 09 de Abril de 2010.

Generating Test Suites for Timed Systems with Context Variables

Adilson Luiz Bonifácio
Computing Institute, University of Campinas
Campinas, Brazil
adilson@ic.unicamp.br

Arnaldo Vieira Moura
Computing Institute, University of Campinas
Campinas, Brazil
arnaldo@ic.unicamp.br

***Abstract* – Model-based formal methods have been widely used for testing critical and reactive systems. Some aspects of reactive systems are captured by the notion of data flow and interactions with the environment. Conventional timed models allow for the continuous evolution of time variables. We propose a new timed context automata to model complex systems, one that simultaneously features time evolution and context variable transformations. We propose a generalized strategy to discretize such models. From the resulting discrete automata we can automatically generate test suites that can be used to verify whether candidate implementations conform to given specifications.**

***keywords* - Timed systems; Discretization; Context transformations; Timed context test case generation;**

1 Introduction

The process of verifying and testing complex computational systems have been intensively investigated. Model-based testing has been one of the most promising techniques among those used to automatically generate test suites for complex systems [1, 2, 3, 4]. Several formalisms have been developed to automatically construct test suites for pure timed systems as well as for systems with only context transformations. An approach to deal with these aspects in the same model has remained elusive.

Here we treat complex systems that obey both time requirements and also satisfy data flow transformations, with both these aspects being resolved within the same formal model. Traditionally, Timed Input/Output Automata (TIOA) [5, 6, 7], a variant of the classical timed automata model [8, 9, 10], have been used to express continuous time evolution. In order to also capture data flow effects, we extend the TIOA model by introducing context variables. The new formalism, Timed Input/Output Context Automata (TIOCA), allow for both continuous time evolution [11, 12] and data flow transformations [13, 14, 15, 16].

In order to extract test suites from these models we propose a new discretization mechanism using the notion of grid automata [11, 17, 18], but now including also data flow transformations. The discretization technique is inspired by ideas proposed in [7], but deviates from the traditional approach based on classical clock regions [8, 19] which treat only the continuous time evolution aspects. Also, notably, the discretization technique now allows for a much wider

range of granularity choices and, furthermore, makes precise the notion of simulation boundaries. With that, it is then possible to precisely establish the relationship between the original system behavior and the corresponding grid automaton, when the former moves away from the established boundaries. More specifically, we show that TIOCA homomorphically simulates their corresponding grid automata, and vice-versa.

In order to specify which system properties will be subjected to testing we use the notion of test purpose models [20, 21]. The joint behavior of a specification and a test purpose is captured by their synchronous product. After discretizing the product, we can automatically extract test suites. Such test suites, when applied to implementation candidates result in complete runs from which test conformance verdicts can be finally obtained [1].

In Section 2, we discuss bounded values and bounded functions. In Section 3, we present the TIOCA model. Section 4 presents the new discretization technique. Section 5 discusses the process of generating test suites using the notions of test purposes and the synchronous product. It also shows how to apply test cases to obtain test verdicts. In Section 6 we briefly survey some related works. Finally, some concluding remarks appear in Section 7.

Full proofs of all claims can be found in [18].

2 Bounds and adjusted values

In this section, we define the notion of bounded values and bounded functions. The latter will be used to limit the excursions of discretized timed automata up to a pre-defined boundary. We also introduce the notion of a set of variables and the corresponding set of rational conditions that can be derived from them. Such conditions will be used to specify both guards along transitions as well as state invariants. In order to keep the number of states of a timed model under control, we will use the notion of discretized automata. Although discretization is introduced in a later section, it will depend on the notion of adjusted values, which is also defined in this section.

2.1 Conditions and interpretations

In what follows, the set of rationals will be denoted by $\mathbb{Q}_{\geq}$. Given $t \in \mathbb{Q}_{\geq}$, we will denote the integral and fractional parts of t by $\lfloor t \rfloor$ and $\lceil t \rceil$, respectively.

Let C be a finite set of symbols, also called variables. A variable interpretation, or simply an interpretation, over C is a partial function from C into $\mathbb{Q}_{\geq}$. A total variable interpretation, or just a total interpretation, over C is a variable interpretation over C whose domain[1] is C. The set of all interpretations and total interpretations over C, respectively, will be denoted by $[C \rightharpoonup \mathbb{Q}_{\geq}]$ and $[C \to \mathbb{Q}_{\geq}]$.

The set of all conditions over variables is defined next.

[1] $\mathrm{dom}(f)$ will denote the domain of a function f.

Definition 1 *Let C be a set of variables. The set of all variable conditions, Φ_C, is comprised by all expressions δ that can be finitely generated using the rules*

$$\delta := \texttt{true} \mid c \leq \tau \mid \tau \leq c \mid \neg\delta \mid \delta_1 \wedge \delta_2,$$

where c is a variable and $\tau \in \mathbb{Q}_{\geq}$. •

We will take the usual liberties when writing variable conditions, *e.g.*, we may write $c \geq \tau$ for $\tau \leq c$, or $c < \tau$ instead of $\neg(\tau \leq c)$, or $\tau_1 \leq c \leq \tau_2$ for $(\tau_1 \leq c) \wedge (c \leq \tau_2)$.

Satisfiability is treated in the usual way.

Definition 2 *Let $\delta \in \Phi_C$ and $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ with all variables occurring in δ in dom(ν). Then ν satisfies δ, denoted $\nu \vDash \delta$, if δ is true when every variable c is replaced by $\nu(c)$ in δ and the resulting propositional logic sentence is evaluated in the usual manner.* •

We now define how to displace an interpretation.

Definition 3 *Let $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ and $\tau \in \mathbb{Q}_{\geq}$. Then $(\nu + \tau)(c) = \nu(c) + \tau$ if $c \in$ dom(ν), or undefined otherwise, for all $c \in C$.* •

That is, $\nu + \tau$ just adds τ to all values under ν. Another important operation is interpretation overruling.

Definition 4 *Let ν, $\mu \in [C \curvearrowright \mathbb{Q}_{\geq}]$. We define $(\nu \oplus \mu)(c)$ as $\mu(c)$ if $c \in$ dom(μ), as $\nu(c)$ if $c \in$ (dom(ν) − dom(μ)), or undefined otherwise, for all $c \in C$.* •

That is, $\nu \oplus \mu$ assigns values first according to μ and, baring that, then it assigns values according to ν, if at all possible.

2.2 Bounded values and functions

We turn to the notion of bounded interpretations.

Definition 5 *Let $L \in \mathbb{Q}_{\geq}$ be a bound. Let $x \in \mathbb{Q}_{\geq}$ be a value, let A be a set and let $\alpha \in [A \curvearrowright \mathbb{Q}_{\geq}]$. Define*

1. *The L-bounded x value, denoted x_L, is given by: (i) x, if $x \leq L$; or (ii) $\lfloor L \rfloor + \lceil x \rceil$, otherwise.*
2. *The L-bounded α function, denoted α_L, is obtained by letting $\alpha_L(a) = (\alpha(a))_L$, for all $a \in A$.* •

We could have specified a different bound L_c for each variable c. In order to keep the notation uncluttered, however, we will consider a single bound L for all variables. It should be a simple matter to generalize any of the following results to the case when some variable bounds may be distinct.

The next two propositions state that the L-bound of a sum of terms is the same as the L-bound of the sum of the L-bounds of the individual terms.

Proposition 6 *Let $L \in \mathbb{Q}_{\geq}$ and let $x_i \in \mathbb{Q}_{\geq}$, for $i = 1, \ldots, n$, with $n \geq 1$. Let $y_i = x_i$ or $y_i = (x_i)_L$, for $i = 1, \ldots, n$. Then*

$$\big(\sum_{i=1}^{n} x_i\big)_L = \big(\sum_{i=1}^{n} y_i\big)_L. \quad \bullet$$

Proposition 7 *Let $L \in \mathbb{Q}_{\geq}$ be a bound and let $W = \{w_1, \ldots, w_n\}$ be a set, with $n \geq 1$. Also let $k_i \geq 0$ be integer constants, $i = 1, \ldots, n$. Take $\alpha \in [W \to \mathbb{Q}_{\geq}]$. Then*

$$\Big[\sum_{i=1}^{n} k_i \alpha(w_i)\Big]_L = \Big[\sum_{i=1}^{n} k_i \alpha_L(w_i)\Big]_L. \quad \bullet$$

The next proposition examines the result of L-bounding displaced and overruled interpretations.

Proposition 8 *Let C be a set of variables, L a positive integer, $\eta \in \mathbb{Q}_{\geq}$ and $\nu, \theta \in [C \to \mathbb{Q}_{\geq}]$. Then $(\nu \oplus \theta)_L = (\nu_L \oplus \theta)_L$ and $(\nu + \eta)_L = (\nu_L + \eta)_L$.* $\bullet$

Function addition takes its usual meaning. The next lemma says that satisfiability of variable conditions is oblivious to L-bounding operands in function additions.

Lemma 9 *Let C be a set of variables, $\alpha_i \in [C \to \mathbb{Q}_{\geq}]$, and $\beta_i = \alpha_i$ or $\beta_i = (\alpha_i)_L$, $i = 1, 2$. Let $I \in \Phi_C$ and let L be a positive integer greater than all constants occurring in I. Then $\alpha_1 + \alpha_2 \vDash I$ iff $\beta_1 + \beta_2 \vDash I$.* $\bullet$

A similar result holds when overruling.

Lemma 10 *Let C be a set of variables, $I \in \Phi_C$, L a positive integer greater than all constants occurring in I and $\nu, \theta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$. If $\nu \oplus \theta \vDash I$ then $\nu_L \oplus \theta \vDash I$.* $\bullet$

2.3 Adjusted values

When discretizing a formal model, we will need to choose proper granularities. In the discretized model, all constants will be *adjusted*, or integer multiples of the granularity. In this section, we treat the notion of adjusted values.

Definition 11 *Let $g \in \mathbb{Q}_{\geq}$ and C a set of variables. A value $x \in \mathbb{Q}_{\geq}$ is* g-adjusted *iff it is an integer multiple of g. A condition $\delta \in \Phi_C$ is* g-adjusted *iff all constants occurring in δ are g-adjusted. An interpretation $\nu \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ is* g-adjusted *iff $\nu(c)$ is g-adjusted, for all $c \in dom(\nu)$.* •

A simple fact about adjusted values follow.

Proposition 12 *Let $g = 1/k$, k a positive integer. Let C be a set of variables. Let $L, \ell \in \mathbb{Q}_{\geq}$ and ν, $\eta \in [C \rightharpoonup \mathbb{Q}_{\geq}]$ be g-adjusted. Then ν_L, $\nu + \ell$ and $\nu \oplus \eta$ are g-adjusted.* •

Any set of L-bounded interpretations is finite, provided that L is properly adjusted.

Lemma 13 *Let C be a set of variables, and let $g = 1/k$ with k a positive integer. Let $R \subseteq [C \rightharpoonup \mathbb{Q}_{\geq}]$ be a set of g-adjusted interpretations, and let $L \in \mathbb{Q}_{\geq}$ be g-adjusted. Let $R_L = \{\nu_L \mid \nu \in R\}$. Then $|R_L| \leq (k\lfloor L \rfloor + k)^{|C|}$.* •

Using a g-adjusted bound L_c for each variable c, the bound would be $|R_L| \leq \prod_{c \in C} (k\lfloor L_c \rfloor + k)$.

3 Timed I/O Context Automata

In this section we introduce the notions of context variables and of input and output parameters [13], extending the classical timed automaton model. Context variables play the role of common variables in ordinary programming. They can enter into expressions and can be assigned to. Input parameters will be used to read in values passed down by the environment. Similarly, values computed by the system can be passed back to the environment.

3.1 An example

A timed automaton that captures the behavior of a simple multimedia system is presented in [17]. The protocol operates by receiving image frames followed by their respective sound tracks, the latter being supposed to arrive within two time units after the preceding image frame. After the arrival of a sound track, the protocol must send an acknowledgment in no more than three time units after the reception of the image frame and no more than two time units after

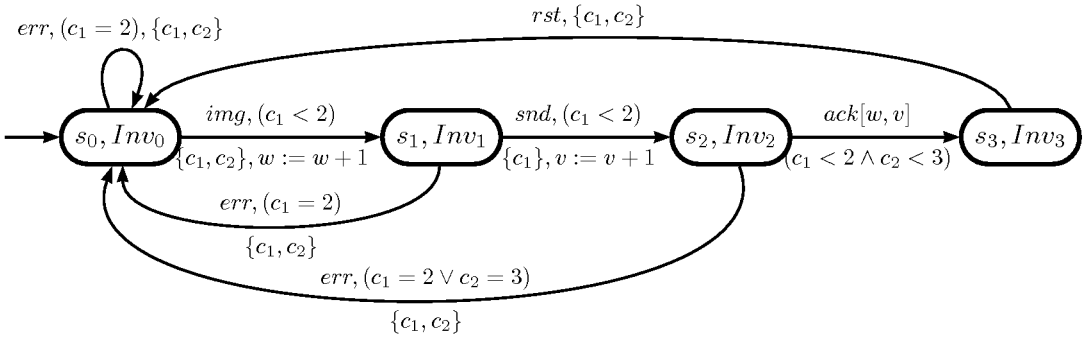


Figure 1: A TIOCA model for a multimedia protocol.

receiving the sound track. A reset message moves the system back to the initial state, so that it can await for the next image frame. Such a message must be issued no more than three time units after the image frame arrives and no more than two time units after the corresponding sound track arrives. When the input sound track does not follow the image frame within two time units, the system times out, issues an error message and goes back to the initial state.

We extended the multimedia protocol, now allowing for context variables, as depicted in Figure 1. The extended model has four states, namely s_0, s_1, s_2 and s_3. The set of clocks is $C = \{c_1, c_2\}$, and the state invariants are $Inv_0 = (c_1 \leq 2)$, $Inv_1 = (c_1 \leq 2)$, $Inv_2 = (c_1 \leq 2 \wedge c_2 \leq 3)$, and $Inv_3 = (c_1 \leq 2 \wedge c_2 \leq 3)$. The set of input actions is $X = \{img, snd, rst\}$, and the set of output actions is $Y = \{err, ack\}$. The intended meaning for the input symbols img and snd is the arrival of an image frame and of a sound track, respectively. The output symbol err signals an error, and the output symbol ack sends an acknowledgement signal back to the environment.

There are seven transitions, indicated by the arrows in the diagram. At each transition, we first indicate the corresponding action symbol. The action symbol may be followed by a list of guards, given within parentheses. The form of the guards pertinent to each type of transition will be detailed shortly. When there are no guards, the list is simply omitted. In the example, at the transition from s_1 to s_2 there is a clock guard in the form $(c_1 < 2)$. At the transition from s_3 to s_0 there are no guards. Next, after the list of guards, the set of clocks to be reset is given between braces. All resets move the corresponding clocks back to zero, with the other clocks remaining unchanged. If there are no clocks to reset, the clock reset list is omitted. In the example, at the transition from s_1 to s_2, only clock c_1 is reset to zero. There are no clock resets in the transition from s_2 to s_3. The last item at each transition indicates the corresponding context variable transformation. In the example, the set of context variables is $V = \{w, v\}$. As we can see from the diagram, at the transition from s_0 to s_1, the context variable transformation is the

expression $w := w + 1$. The intended meaning of such an expression, besides the assignment, is that the other context variable, v, remains unchanged when this transition is taken. When all context variable transformations are omitted, as in the transition from s_2 to s_0, it should be understood that all variables remain unchanged when that transition is taken.

In this simple example, there are no input parameters. There are four transitions associated with some output action symbol in $Y = \{err, ack\}$. The transitions associated with err send no values back to the environment. In the transition on ack, from s_2 to s_3, the notation $ack[w, v]$ indicates that the current values of the variables w and v are passed back to the environment. In the sequel, we give more details about the use of input parameters and values passed back to the environment.

3.2 The extended model

Timed I/O automata are defined in [7]. In addition, we need a finite set R of input parameters, or parameters for short. Let X and Y be the set of input and output symbols, respectively. Each input action symbol will have a particular set of parameters associated to it. We denote by $R_x \subseteq R$ the set of parameters associated with input symbol x, for all $x \in X$. Each parameter will have a value from $\mathbb{Q}_{\geq}$ associated to it, given by a parameter valuation in $[R_x \to \mathbb{Q}_{\geq}]$. The set of R_x conditions will be denoted simply by Φ_x. Recall Definitions 1 and 2. Returning to the example at Section 3.1, we have $X = \{img, snd, rst\}$, $Y = \{err, ack\}$ and $R = \emptyset$.

Proceeding, we consider a set V of context variables, all of which take values in $\mathbb{Q}_{\geq}$. Again, from Definitions 1 and 2 we obtain the sets of context variable valuations $[V \curvearrowright \mathbb{Q}_{\geq}]$ and $[V \to \mathbb{Q}_{\geq}]$, together with the set of context variable conditions Φ_V and the notion of satisfiability $\lambda \vDash \delta$, for a context variable valuation λ and a context variable condition δ. In the example, the set of context variables is $V = \{w, v\}$.

When taking a discrete transition on an input symbol x, the model can also redefine the context variable values according to an expression involving input parameter values associated to x and the current values of the context variables. In order to capture this effect, we associate with each discrete transition on an input symbol x a map that takes a pair (ρ, λ), where $\rho \in [R_x \to \mathbb{Q}_{\geq}]$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$, and returns another context variable valuation in $[V \to \mathbb{Q}_{\geq}]$. We define the set of all such mappings thus

$$F_x = \Big\{ \kappa \,|\, \kappa : [R_x \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}] \Big\}.$$

In the example, since there are no input parameters associated to img, we get $R_{img} = \emptyset$. Since $V = \{w, v\}$, we obtain

$$F_{img} = \Big\{ \kappa \,|\, \kappa : \{\emptyset\} \times [\{w, v\} \to \mathbb{Q}_{\geq}] \to [\{w, v\} \to \mathbb{Q}_{\geq}] \Big\}.$$

Similarly, $F_{snd} = F_{rst} = F_{img}$.

Collecting, a discrete transition over an input action symbol x moves the machine from a state s to a state r, provided that certain guard conditions over clocks, input parameters and context variables are satisfied. Also, the machine can reset some clock values and redefine some context variable values. The new clock values are constants specified directly in the transition. The new context variable values are computed from the input parameter values and from the current values of the context variables. We can express these conditions by saying that a discrete transition over an input action symbol x is a member of the set

$$S \times \{x\} \times \Phi_C \times \Phi_x \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times F_x \times S.$$

As an illustration, take the transition from s_0 to s_1, over the input action symbol img in Figure 1. The clock guard is $(c_1 < 2)$ and the input parameter guard is `true`, which is not listed. The context variable guard is also not listed and so it is also taken as `true`. The set $\{c_1, c_2\}$ says that both clocks are reset to zero when the transition is taken. This can be specified by a mapping $\nu \in [C \curvearrowright \mathbb{Q}_{\geq}]$ where $\nu(c_1) = \nu(c_2) = 0$. Finally, the new value of the context variable w is defined by the expression $w := w+1$. An appropriate mapping κ for this transition has the form

$$\kappa : \{\emptyset\} \times \big[\{w, v\} \to \mathbb{Q}_{\geq}\big] \to \big[\{w, v\} \to \mathbb{Q}_{\geq}\big],$$

where $\kappa(\emptyset, \lambda)(w) = \lambda(w) + 1$ and $\kappa(\emptyset, \lambda)(v) = \lambda(v)$.

We have an analogous situation with discrete transitions over output action symbols. The only difference is that, now, some context variables have their values returned back to the environment. In general, if the current context variable valuation is λ, the machine will output a partial valuation given by $\xi(\lambda)$, where ξ is a mapping specified in the transition. The set of such mappings is defined thus

$$H_y^P = \big\{\xi \,|\, \xi \in [V \to \mathbb{Q}_{\geq}] \to [V \curvearrowright \mathbb{Q}_{\geq}]\big\},$$

where y is the output symbol. In the example, the transition from s_2 to s_3 specifies the output symbol as $ack[w, v]$ indicating that the values of the context variables w and v will be returned unmodified to the environment. In this case, the output mapping ξ would be just the identity, that is, $\xi(\lambda) = \lambda$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$. In the transition from s_1 to s_0, the output symbol is written as err, and we do not pass values to the environment. In this case, $\xi(\lambda) = \emptyset$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$. Also, when an output transition is taken, the values of the context variables may be modified. We model this by a mapping χ that takes a total context variable valuation and returns another total context variable valuation. That is, we specify a mapping from the set

$$H_y^T = \big\{\chi \,|\, \chi \in [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\big\},$$

where y is the output symbol. In the example, no output transitions modify the values of context variables, and so $\chi(\lambda) = \lambda$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$, at all output transitions.

Putting it all together, we say that on a discrete transition over an output action symbol y the machine moves from a state s to a state r, provided that certain clock and context variable conditions are satisfied. Upon taking the transition, clocks may be reset, context variable values are returned and may have their values modified. The clock reset values are constants specified directly at the transition. The values to be returned and the new context variable values are specified by choosing specific transformation mappings. That is, a discrete transition over an output action y is a member of

$$S \times \{y\} \times \Phi_C \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times H_y^P \times H_y^T \times S.$$

We can now define the new timed I/O context automaton.

Definition 14 *A Timed I/O Context Automaton M is given by a tuple $(S, s_0, \Sigma, C, \nu_0, Inv, R, V, \lambda_0, T_X, T_Y)$, where S is the set of states, $s_0 \in S$ is the initial state, $\Sigma = X \cup Y$ is the alphabet with $X \cap Y = \emptyset$, C is the set of clocks, $\nu_0 \in [C \to \mathbb{Q}_{\geq}]$ is the initial clock valuation with $\nu_0(c) = 0$ for all $c \in C$, and $Inv \in [S \to \Phi_C]$ gives the invariant at each state. The set of parameters is R, the set of context variables is V and $\lambda_0 \in [V \to \mathbb{Q}_{\geq}]$ is the initial context variable valuation, where $\lambda_0(v) = 0$ for all $v \in V$. The set of transitions over input actions satisfies*

$$T_X \subseteq \bigcup_{x \in X} (S \times \{x\} \times \Phi_C \times \Phi_x \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times F_x \times S),$$

where F_x is the set of context update of x, given by

$$F_x = \{\kappa \mid \kappa : [R_x \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\},$$

and $R_x \subseteq R$ is the set of input parameters associated to x. Finally, the set of transitions over output actions satisfies

$$T_Y \subseteq \bigcup_{y \in Y} (S \times \{y\} \times \Phi_C \times \Phi_V \times [C \curvearrowright \mathbb{Q}_{\geq}] \times H_y^P \times H_y^T \times S),$$

where $H_y^P = \{\xi \mid \xi \in [V \to \mathbb{Q}_{\geq}] \to [V \curvearrowright \mathbb{Q}_{\geq}]\}$, and $H_y^T = \big\{\chi \mid \chi \in [V \to \mathbb{Q}_{\geq}] \to [V \to \mathbb{Q}_{\geq}]\big\}$, all $y \in Y$. •

A transition in T_X is a tuple $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r)$ saying that the machine can move from state s to state r over the input symbol x provided that the guards δ_1, δ_2, and δ_3, over clock variables, input parameters and context variables, respectively, are all enabled. Further, upon moving to state r, the mapping $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$ indicates which clocks are reset and to which values. Finally, $\kappa \in F_x$ denotes the context variable update function, mapping input parameter

valuations and context variable valuations into new context variable valuations. A transition in T_Y is a tuple $(s, y, \delta_1, \delta_2, \theta, \xi, \chi, r)$ specifying that the machine can move from state s to state r over the output action symbol y, provided that the guards δ_1 and δ_2, over clock variables and context variables, respectively, are enabled. Further, upon moving to state r the mapping $\theta \in [C \curvearrowright \mathbb{Q}_{\geq}]$ gives which clocks are reset and to which values. Lastly, $\xi \in H_y^P$ specifies values to be passed to the environment, and $\chi \in H_y^T$ returns the new values of the context variables.

In order to specify successive moves of a TIOCA we need the notion of timed words.

Definition 15 *Let M be a TIOCA and let $\Upsilon \subseteq \Sigma$. Then $\Psi_\Upsilon = \{(x, \rho) \mid x \in X \cap \Upsilon,\ \rho \in [R_x \to \mathbb{Q}_{\geq}]\} \bigcup \{(y, \rho) \mid y \in Y \cap \Upsilon,\ \rho \in [V \curvearrowright \mathbb{Q}_{\geq}]\}$, is the set of all actions over Υ. A timed word for M is a sequence $\psi = \langle \sigma_1, \ldots, \sigma_n \rangle$, where $n \geq 0$ and $\sigma_i \in \Psi_\Sigma \cup \mathbb{Q}_{\geq}$, $1 \leq i \leq n$.* •

In particular, Ψ_X and Ψ_Y are the sets of input and output action valuations, respectively, of M. We denote the set of all timed words for M by Ψ_M. The empty timed word will be denoted by ε. The concatenation of timed words follows the standard string concatenation definition.

A configuration for a TIOCA M is a triple (s, ν, λ), where $s \in S$, $\nu \in [C \to \mathbb{Q}_{\geq}]$ and $\lambda \in [V \to \mathbb{Q}_{\geq}]$. In the initial configuration, s is the initial state, ν is the initial clock interpretation and λ is the initial context valuation of M. Let Γ_M denote the set of configurations of M.

We can now define the semantics of a TIOCA.

Definition 16 *Let M be a TIOCA and let $\gamma_i = (s_i, \nu_i, \lambda_i) \in \Gamma_M$, $i = 1, 2$, be two configurations of M.*

1. *Let $\tau \in \mathbb{Q}_{\geq}$. There exists a* continuous move *from γ_1 to γ_2 over τ, denoted by $\gamma_1 \underset{\tau}{\to} \gamma_2$, if and only if: (i) $s_1 = s_2$; (ii) $\nu_2 = \nu_1 + \tau$; (iii) $\nu_1 + \eta \vDash Inv(s_1)$ for all η, $0 < \eta \leq \tau$; and (iv) $\lambda_2 = \lambda_1$.*

2. *Let $(x, \rho) \in \Psi_X$. There is a* discrete move *from γ_1 to γ_2 over (x, ρ), iff there is $(s_1, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, s_2) \in T_X$ such that: (i) $\nu_1 \vDash \delta_1$, $\rho \vDash \delta_2$ and $\lambda_1 \vDash \delta_3$; (ii) $\nu_2 = \nu_1 \oplus \theta$ and $\nu_2 \vDash Inv(s_2)$; and (iii) $\lambda_2 = \kappa(\rho, \lambda_1)$. We denote this move by $\gamma_1 \overset{(x,\rho)}{\to} \gamma_2$.*

3. *Let $(y, \mu) \in \Psi_Y$. There is a* discrete move *from γ_1 to γ_2 over (y, μ), iff there is $(s_1, y, \delta_1, \delta_2, \theta, \xi, \chi, s_2) \in T_Y$ such that: (i) $\nu_1 \vDash \delta_1$ and $\lambda_1 \vDash \delta_2$; (ii) $\nu_2 = \nu_1 \oplus \theta$ and $\nu_2 \vDash Inv(s_2)$; and (iii) $\mu = \xi(\lambda_1)$ and $\lambda_2 = \chi(\lambda_1)$. We denote this move by $\gamma_1 \overset{(y,\mu)}{\to} \gamma_2$.* •

Decorations over and under $\longrightarrow$ may be dropped.

The movements of a TIOCA can now be specified.

Definition 17 *Let M be a TIOCA. The* movement relation *of M, $\vdash_M \subseteq (\Psi_M \times \Gamma_M)$ is such that $(\langle\sigma\rangle \cdot \psi, \gamma_1) \vdash_M (\psi, \gamma_2)$, with $\sigma \in (\Psi_M \cup \mathbb{Q}_{\geq})$ and $\psi \in \Psi_M$, iff either (i) $\sigma \in \mathbb{Q}_{\geq}$ and $\gamma_1 \xrightarrow[\sigma]{} \gamma_2$; or (ii) $\sigma \in \Psi_\Sigma$ with $\gamma_1 \xrightarrow{\sigma} \gamma_2$.* •

When $(\psi, \gamma) \vdash_M (\varepsilon, \varphi)$ we also write $\gamma \Vdash_M^{\psi} \varphi$, or $\gamma \Vdash_M \varphi$ when ψ is not relevant. When $\gamma \Vdash_M^{\psi} \varphi$ we say that we have a *run starting* at γ and *ending* at φ over ψ. A configuration γ is *reachable* if there is an execution ending at γ. Also, a state s is *reachable* if there is a reachable configuration (s, ν, λ), for some clock interpretation ν and some context variable valuation λ.

4 TIOCA discretization

In order to obtain TIOCA discretizations, we need to discretize both clock and context variable values.

4.1 Clock and context variable valuations

From now on, $L, K \in \mathbb{Q}_{\geq}$ are the clock and context variable boundaries, respectively. Recall Definition 5. We could have used distinct L_c, K_v for $c \in C$ and $v \in V$, but it would only overload the notation. It is easy to generalize the next results when distinct L_c and K_v are used.

Mappings in H_y^P and in H_y^T will obey a linear form.

Definition 18 *Let V be a set of variables and let ξ map $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. Then ξ is* fully linear *iff for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$ and $v \in dom(\xi(\lambda))$ there is an integer constant b, and a set of integer constants $\{a_w \mid w \in V\}$ with*

$$\xi(\lambda)(v) = \Big[\sum_{w \in V} a_w \lambda(w)\Big] + b. \bullet$$

Mappings $\kappa \in F_x$ will obey a bi-linearity condition.

Definition 19 *Let V be a set of variables and R a set of parameters. Let κ map $[R \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. Then κ is* fully bi-linear *iff there are fully linear mappings ξ_1, from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, and ξ_2 from $[R \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$, such that $\kappa(\rho, \lambda) = \xi_1(\lambda) + \xi_2(\rho)$, for all $\rho \in [R \to \mathbb{Q}_{\geq}]$ and all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.* •

The next facts expose the behavior of linear mappings.

Fact 20 *Let V be a set of variables. Let ξ be a fully linear mapping from $[V \to \mathbb{Q}_{\geq}]$ into $[V \rightharpoonup \mathbb{Q}_{\geq}]$. Then $\xi_K(\lambda) = \xi_K(\lambda_K)$, for all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.* •

Fact 21 *Let V be a set of variables, R a set of parameters and κ a fully bi-linear mapping from $[R \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ into $[V \curvearrowright \mathbb{Q}_{\geq}]$. Then $\kappa_K(\rho, \lambda) = \kappa_K(\rho_K, \lambda_K)$, for all $\rho \in [R \to \mathbb{Q}_{\geq}]$ and all $\lambda \in [V \to \mathbb{Q}_{\geq}]$.* •

We adopt the following assumption.

Assumption 22 *Let M be a TIOCA. All mappings in F_x are fully bi-linear, for all $x \in X$, and all mappings in $H_y^P \cup H_y^T$ are fully linear, for all $y \in Y$.* •

4.2 Grid automata and TIOCA

First, we fix some notation. Recall Definition 11.

Assumption 23 *Clock and context grid units will be in the forms $g = 1/k$ and $h = 1/\ell$, with k and ℓ positive integers, respectively.* •

A TIOCA is adjusted when all constants occurring in its definition are properly adjusted.

Definition 24 *A TIOCA M is* $[g, h]$-adjusted *iff for all $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r) \in T_X$, $(s, y, \delta_1, \delta_3, \theta, \xi, \chi, r) \in T_Y$ we have that δ_1, θ are g-adjusted, and δ_2, δ_3 are h-adjusted. Further, for all $s \in S$, $Inv(s)$ is g-adjusted.* •

Applying update functions over h-adjusted parameter and context valuations always results in h-adjusted valuations.

Proposition 25 *If M is a $[g, h]$-adjusted TIOCA then $\xi(\lambda)$ and $\kappa(\rho, \lambda)$ are h-adjusted, all $y \in Y$, $x \in X$, $\kappa \in F_x$, $\xi \in H_y^P \cup H_y^T$, $\lambda \in [V \to \mathbb{Q}_{\geq}]$ and $\rho \in [R_x \to \mathbb{Q}_{\geq}]$.* •

A timed word is adjusted if all its time instants and all its parameter values are properly adjusted.

Definition 26 *A timed word $\langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ is $[g, h]$-adjusted iff σ_i is g-adjusted when $\sigma_i \in \mathbb{Q}_{\geq}$, and ρ is h-adjusted when $\sigma_i = (z, \rho) \in \Psi_\Sigma$, all $i \in \{1, \ldots, n\}$.* •

The set of all $[g, h]$-adjusted timed words will be denoted by $\Psi_{[g,h]}$.

The $[g, h]$-reachability of configurations is defined next.

Definition 27 *Let M be a $[g, h]$-adjusted TIOCA, $s \in S$, $\nu \in [C \to \mathbb{Q}_{\geq}]$, $\lambda \in [V \to \mathbb{Q}_{\geq}]$. Then (s, ν, λ) is $[g, h]$-reachable iff $(s_0, \nu_0, \lambda_0) \overset{\psi}{\Vdash}_M (s, \nu, \lambda)$ with $\psi \in \Psi_{[g,h]}$.* •

Now we can define the grid corresponding to a TIOCA.

Input: L, $K \in \mathbb{Q}_{\geq}$; k, ℓ positive integers, $g = 1/k$, $h = 1/\ell$; $[g,h]$-adjusted TIOCA M.
Output: Grid $M_G = (S_G, s_G, \Sigma_G, T_G)$.

let $\Psi_z^K = \{(z, \rho_K) | (z,\rho) \in \Psi_{\{z\}}\}$, $z \in \Sigma$;
begin
  let $s_G = (s, \nu_0, \lambda_0)$; $T_G \leftarrow \emptyset$; $HS \leftarrow \emptyset$; $RS \leftarrow s_G$;
  while $RS \setminus HS \neq \emptyset$ **do**
    get a (s,ν,λ) from $RS \setminus HS$; **move** (s,ν,λ) to HS;
    foreach $(s, x, \delta_1, \delta_2, \delta_3, \theta, \kappa, r) \in T_X$ **do**
      if $\nu \vDash \delta_1$ *and* $\lambda \vDash \delta_3$ *and* $\nu \oplus \theta \vDash Inv(r)$ **then**
        let $\eta = (\nu \oplus \theta)_L$;
        foreach (x, ρ) **in** Ψ_x^K **do**
          if $\rho \vDash \delta_2$ **then**
            let $\mu = \kappa_K(\rho, \lambda)$
            add $((r,\nu,\lambda),(x,\rho),(r,\eta,\mu))$ to T_G;
            add (r,η,μ) to RS, if $(r,\eta,\mu) \notin HS$;
          end
        end
      end
    end
    foreach $(s, y, \delta_1, \delta_2, \theta, \xi, \chi, r) \in T_Y$ **do**
      if $\nu \vDash \delta_1$ *and* $\lambda \vDash \delta_2$ *and* $\nu \oplus \theta \vDash Inv(r)$ **then**
        let $\eta = (\nu \oplus \theta)_L$; $\rho = \xi_K(\lambda)$; $\mu = \chi_K(\lambda)$;
        add $((s,\nu,\lambda),(y,\rho),(r,\eta,\mu)$ to T_G;
        add (r,η,μ) to RS, if $(r,\eta,\lambda) \notin HS$;
      end
    end
    if $\nu + h \vDash Inv(s)$ *for all* $0 < h \leq g$ **then**
      let $\eta = (\nu + g)_L$;
      add $((s,\nu,\lambda), g, (s,\eta,\lambda))$ to T_G;
      add e (s,η,λ) to RS, if $(s,\eta,\lambda) \notin HS$;
    end
  end
  $S_G \leftarrow HS, \quad \Sigma_G \leftarrow \{g\} \cup \bigcup_{z \in \Sigma} \Psi_z^K$;
  return;
end

Algorithm 5: Grid algorithm.

Definition 28 *Let M be a $[g,h]$-adjusted TIOCA. The grid automaton associated with M is the labelled transition system constructed by Algorithm 5.* •

Algorithm 5 always terminates and there is a bound on the number of states in the labelled transition system.

Lemma 29 *Algorithm 5 halts with $|S_G| \leq |S| \times \big(k\lfloor L \rfloor + k\big)^{|C|} \times \big(\ell\lfloor K \rfloor + \ell\big)^{|V|}$.* •

Again, if we had different L_c and K_v values, the number of states in the grid would be bounded by $|S| \times \prod_{c\in C} (k\lfloor L_c \rfloor + k) \times \prod_{v\in V} (\ell\lfloor K_v \rfloor + \ell)$, which can be much smaller then $|S| \times \big(k\lfloor L \rfloor + k\big)^{|C|} \times \big(\ell\lfloor K \rfloor + \ell\big)^{|V|}$, if we take the safe values $L = \max_{c\in C}\{L_c\}$ and $K = \max_{v\in V}\{L_v\}$.

Now, we want to argue that a TIOCA and its grid display compatible behaviors. A grid word in $\Sigma_G^\star$ induces a movement in the grid automaton in a usual way. Remember that a grid word carries the symbol g, as well as h-adjusted valuations $(z,\rho) \in \Psi_\Sigma$.

Definition 30 *Let M_G be the grid automaton corresponding to a TIOCA M. The* movement relation *of M_G, $\vdash_G$, is a binary relation over $\Sigma_G^\star \times S_G$ given by $(\langle\sigma\rangle\psi, s) \vdash_G (\psi, r)$ if and only if there is a transition (s,σ,r) in M_G.* •

We may write $\gamma \overset{\psi}{\Vdash}_G \rho$ instead of $(\psi,\gamma) \overset{\star}{\vdash}_G (\varepsilon,\rho)$, and $\gamma \Vdash_G \rho$ when ψ is not relevant.

4.3 Relationship between a TIOCA and its grid

All $[g,h]$-reachable configurations are states in the grid.

Lemma 31 *M is a TIOCA and M_G its associated grid. Let L, K be positive integers greater than any constant occurring in M. If $(s,\nu,\lambda) \in S \times [C \to \mathbb{Q}_{\geq}] \times [V \to \mathbb{Q}_{\geq}]$ is $[g,h]$-reachable in M then $(s,\nu_L,\lambda_K) \in S_G$.* •

Grid states correspond to reachable configurations.

Lemma 32 *Let M_G be the grid corresponding to a TIOCA M. Let L, K be positive integers greater than any constant occurring in M. If $(s,\nu,\lambda) \in S_G$ then there are $\mu \in [C \to \mathbb{Q}_{\geq}]$, $\omega \in [V \to \mathbb{Q}_{\geq}]$ and a $[g,h]$-adjusted parameterized timed word $\psi \in \Psi_{[g,h]}$ such that $(s_0,\nu_0,\lambda_0) \overset{\psi}{\Vdash}_M (s,\mu,\omega)$, with $\mu_L = \nu$ and $\omega_K = \lambda$.* •

The next definition maps adjusted timed words into grid words. Recall Definition 26.

Definition 33 *Let M be a TIOCA. Define the basic mappings: (i) every g-adjusted time value $ig \in \mathbb{Q}_{\geq}$, with $i \geq 0$, is mapped to the grid word $g^i \in \Sigma_G^\star$; and (ii) every symbol $(z, \mu) \in \Psi_\Sigma$ is mapped to (z, μ_K). Define the morphism $f : \Psi_{[g,h]} \to \Sigma_G^\star$ by extending these basic mappings in the usual way.* •

Now, a grid simulates the corresponding TIOCA.

Theorem 34 *M is a TIOCA. Let $L, K \in \mathbb{Q}_{\geq}$ be greater than all constants occurring in M, and let M_G be the grid of M. Let $s, r \in S$, $\nu, \omega \in [C \to \mathbb{Q}_{\geq}]$ and $\lambda, \mu \in [V \to \mathbb{Q}_{\geq}]$ be such that (s, ν, λ) is $[g, h]$-reachable in M. If $(s, \nu, \lambda) \overset{\psi}{\Vdash}_M (r, \omega, \mu)$ then $(s, \nu_L, \lambda_K), (r, \omega_L, \mu_K) \in S_G$ and $(s, \nu_L, \lambda_K) \overset{f(\psi)}{\underset{G}{\Vdash}} (r, \omega_L, \mu_K)$.* •

And also a TIOCA imitates the corresponding grid.

Theorem 35 *M_G is the grid of a TIOCA M. Let $L, K \in \mathbb{Q}_{\geq}$ be greater than all constants occurring in M. Let $(s, \widehat{\omega}, \widehat{\mu}), (r, \omega, \mu) \in S_G$ with $(s, \widehat{\omega}, \widehat{\mu}) \overset{\psi}{\underset{G}{\Vdash}} (r, \omega, \mu)$, some $\psi \in \Sigma_G^\star$. Then there are $\alpha, \widehat{\alpha} \in [C \to \mathbb{Q}_{\geq}]$, $\beta, \widehat{\beta} \in [V \to \mathbb{Q}_{\geq}]$ and $\widehat{\psi} \in \Psi_{[g,h]}$, such that $(s, \widehat{\alpha}, \widehat{\beta}) \overset{\widehat{\psi}}{\Vdash}_M (r, \alpha, \beta)$, with $\widehat{\omega} = \widehat{\alpha}_L$, $\omega = \alpha_L$, $\widehat{\mu} = \widehat{\beta}_K$, $\mu = \beta_K$ and $f(\widehat{\psi}) = \psi$.* •

5 Generating test cases

We want to generate test cases using test purposes.

Definition 36 *A TIOCA is acyclic iff the subjacent graph defined by its states as nodes and transitions as edges is acyclic. A test purpose is an acyclic TIOCA with fail states $F \subseteq S$ and desired states $D \subseteq S$, with $F \cap D = \emptyset$.* •

We construct the synchronous product algorithmically.

Definition 37 *Let M_1, M_2 be TIOCA, with $C_1 \cap C_2 = \emptyset$, and $V_1 \cap V_2 = \emptyset$. The synchronous product of M_1 and M_2 is constructed by Algorithm 6.* •

Algorithm 6 first pairs the initial states of both TIOCA to get the initial state of the product. Product transitions are obtained by searching down for new states and transitions.

A state (s_1, s_2) is a fail state in the product when s_2 is a fail state in the purpose model. Similarly for desired states.

Definition 38 *Let M_1 be a TIOCA and M_2 a test purpose. In Definition 37, (s_1, s_2) is a desired or fail state iff s_2 is a desired or fail state, respectively, in M_2.* •

Input: TIOCA M_1, M_2 with $C_1 \cap C_2 = \emptyset$, $V_1 \cap V_2 = \emptyset$.
Output: The synchronous product M_P.
begin
  $C_P \leftarrow C_1 \cup C_2$; $R_P \leftarrow R_1 \cup R_2$; $V_P \leftarrow V_1 \cup V_2$;
  $\Sigma_P \leftarrow \Sigma_1 \cup \Sigma_2$; $RS \leftarrow s_0^P = (s_0^1, s_0^2)$; $Inv_P(s_0^P) \leftarrow Inv_1(s_0^1) \wedge Inv_2(s_0^2)$; $T_P \leftarrow \emptyset$, $HS \leftarrow \emptyset$;
  while $RS \setminus HS \neq \emptyset$ **do**
    choose $s = (s_1, s_2)$ from $RS \setminus HS$;
    move s from RS to HS;
    foreach $a \in X$ **do**
      if $(s_i, a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, s_{i+2}) \in T_i^X$, $i = 1, 2$ **then**
        add (s_3, s_4) to RS;
        let $Inv_P(s_3, s_4) \leftarrow Inv_1(s_3) \wedge Inv_2(s_4)$;
        add $((s_1, s_2), a, \delta_1^1 \wedge \delta_2^1, \delta_1^2 \wedge \delta_2^2, \delta_1^3 \wedge \delta_2^3, \theta_1 \oplus \theta_2, \kappa_1 \oplus \kappa_2, (s_3, s_4))$ to T_P^X;
      end
      if $(s_i, a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, s_{i+2}) \in T_i^X$ *for some* $s_{i+2} \in S$, *and* $(s_j, a, \delta_j^1, \delta_j^2, \delta_j^3, \theta_j, \kappa_j, s_{j+2}) \notin T_j^X$ *for all* $s_{j+2} \in S$, *with* $i \neq j$, $i, j \in \{1, 2\}$ **then**
        if $i = 1$ **then** $(p, q) = (s_3, s_2)$
        else $(p, q) = (s_1, s_4)$;
        add (p, q) to RS;
        let $Inv_P(p, q) \leftarrow Inv_1(p) \wedge Inv_2(q)$;
        add $((s_1, s_2), a, \delta_i^1, \delta_i^2, \delta_i^3, \theta_i, \kappa_i, (p, q))$ to T_P^X;
      end
    end
    foreach $a \in Y$ **do**
      if $(s_i, a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi, s_{i+2}) \in T_i^Y$, $i = 1, 2$ **then**
        add (s_3, s_4) to RS;
        let $Inv_P(s_3, s_4) \leftarrow Inv_1(s_3) \wedge Inv_2(s_4)$;
        add $((s_1, s_2), a, \delta_1^1 \wedge \delta_2^1, \delta_1^2 \wedge \delta_2^2, \theta_1 \oplus \theta_2, \xi_1 \oplus \xi_2, \chi_1 \oplus \chi_2, (s_3, s_4))$ to T_P^Y;
      end
      if $(s_i, a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi_i, s_{i+2}) \in T_i^Y$ *for some* $s_{i+2} \in S$, *and* $(s_j, a, \delta_j^1, \delta_j^2, \theta_j, \xi_j, \chi_j, s_{j+2}) \notin T_j^Y$ *for all* $s_{j+2} \in S$, *with* $i \neq j$, $i, j \in \{1, 2\}$, **then**
        if $i = 1$ **then** $(p, q) = (s_3, s_2)$ **else** $(p, q) = (s_1, s_4)$;
        add (p, q) to RS;
        let $Inv_P(p, q) \leftarrow Inv_1(p) \wedge Inv_2(q)$;
        add $((s_1, s_2), a, \delta_i^1, \delta_i^2, \theta_i, \xi_i, \chi_i, (p, q))$ to T_P^Y;
      end
    end
  end
  $S_P \leftarrow HS$;
end

Algorithm 6: Synchronous product for TIOCA.

After the grid construction, test sequences are extracted by traversing the grid. The traversal operation starts at the initial state of the grid and searches down until it finds fail or desired states, depending on the nature of the test. Upon finding one such state, the corresponding test sequence is output. A recursive traversal procedure is depicted in Algorithm 7. Clearly, the set of all test sequences is generated by a call in the form $TiocaTestGeneration(s_0, \epsilon)$, where s_0 is the initial state of the grid and ϵ is the empty string.

```
1  TiocaTestGeneration(INPUT: state s of a TIOCA grid M_G; OUTPUT:
   Parameterized timed test sequence PTTS;)
2  begin
3      if s is a leaf then
4          Write PTTS;
5      else
6          foreach neighbor, r, of s reached over a transition on σ do
7              PTTS ← {σ} · PTTS;
8              TiocaTestGeneration(r,PTTS);
9          end
10     end
11 end
```

Algorithm 7: The traversal algorithm for TIOCA.

5.1 Test case generation and the grid automaton

The grid automaton is obtained from the resulting product by applying the method of Section 4. After the grid construction, test sequences are extracted by recursively traversing it. The traversal starts at the initial state and searches down until it finds fail or desired states, depending on the nature of the test. Upon finding one such state, the corresponding path from the initial state is output as a test sequence, the recursion returns one level and starts down another branch. One can then construct $[g, h]$-adjusted timed words from all grid words extracted from the grid automaton using the mapping given at Definition 33.

In order to drive a TIOCA, we supply it with a sequence of time delays and input symbols. In order to extract its behavior, we need to project timed word onto the set of output actions.

Definition 39 *Let Σ be an alphabet and $\Upsilon \subseteq \Sigma$. Let $\psi = \langle \sigma_1, \ldots, \sigma_n \rangle$, $n \geq 0$, be a timed word over Σ. The projection of ψ over Υ, denoted $\psi{\downarrow_\Upsilon}$, is the timed word over Υ given by $\psi{\downarrow_\Upsilon} = \phi_1 \cdot \phi_2 \cdot \ldots \cdot \phi_n$, where $\phi_i = \langle \sigma_i \rangle \cup \mathbb{Q}_{\geq}$ if $\sigma_i \in \Psi_\Upsilon \cup \mathbb{Q}_{\geq}$, or ε otherwise.* •

That is, the projection extracts only the actions in the subset of interest, together with timing values.

Test cases are timed words where only input actions occur.

Definition 40 *A* test sequence *for a TIOCA M is any timed word ψ such that $\psi = \psi{\downarrow_X}$.* •

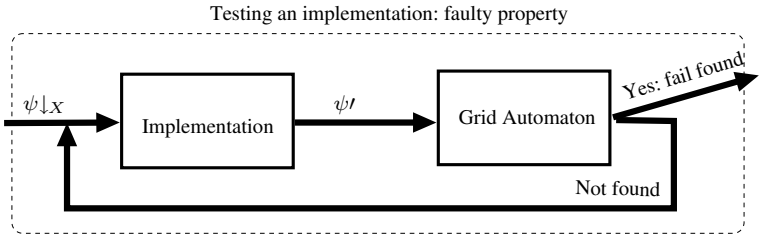


Figure 2: The framework to test implementations: faulty property.

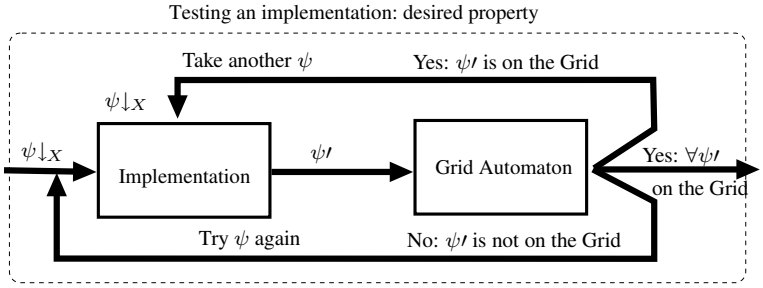


Figure 3: The framework to test implementations: desired property.

5.2 Applying test cases

An implementation behavior is investigated by applying stimuli to it and observing its responses [1].

We obtain a set of test *sequences* by traversing the corresponding grid automaton which, in turn, was obtained from the product between the specification and the test purpose. Such test sequences are sequences of input and output actions, as well as time delays. Test *cases* are then extracted from the test sequences by projecting the latter onto input actions.

A test execution is obtained by applying a test case to an implementation and observing its responses. The outputs of the implementation are then combined with the respective test case. By this process, we obtain test executions, also called *runs*, which combine input actions and time delays from the test cases, together with time delays and observed outputs from the implementation.

Suppose that a test case is submitted to an implementation under test. Then, usually, a time delay must pass before the next input action occurs, or before the next output action is observed. Note that the time delay preceding a output symbol is not under the control of the observer, and so we cannot know in advance the exact instants when outputs will occur. In any case, when collecting a test run, the time delay actually observed before an output symbol is adjoined to the run being constructed, followed by the corresponding output symbol. When the next action is

an input symbol, the time delay specified on the test case is under control of the observer. This time delay, followed by the input symbol, is also adjoined to the run being collected. So, a set of runs can be collected by applying test cases over the candidate implementation. After that, we apply such runs again to the corresponding grid automaton. We can, then, verify if runs obtained from the implementation candidate are also runs over the corresponding grid automaton.

When a faulty property is specified by the test purpose model, the product automaton will contain special faulty states. So, the test cases extracted from the grid will be sequences of input actions and time delays that lead to faulty states. We then collect runs over the implementation using these test cases. When applying such runs back to the original grid, if any of them reaches a faulty state, we can announce a positive test verdict, that is, a fault was confirmed over the implementation. If all runs do not reach faulty states, then the test is deemed inconclusive. See Figure 2.

When the test purpose models a desired property, we must make sure that all runs applied to the grid terminate at desired states. In this case, the test is positive, otherwise it is inconclusive. See Figure 3.

Note that, in this approach, we test implementations by resubmitting runs back to the grid automaton. If we assume that the grid automaton is fully constructed and is kept in memory, then we can efficiently submit a run to the grid. As a second alternative, avoiding the full grid construction, we can use the grid construction algorithm to resubmit collected runs to the grid. When dealing with fault properties, avoiding the full grid construction can be a reasonable alternative, since we only need one hit.

6 Related works

Many works have discussed formal methods to automatically generate test suites for complex systems. Depending on specific characteristics of the target systems we can find either techniques that deal only with time, or methods applicable to systems with context variables and no time relationships. Now we briefly describe some such works.

En-Nouaary and Dssouli discuss a test case generation method in [11]. This method is based on timed automata specifications and test purposes, and uses the notion of synchronous product. However, their discretization technique is based on the classical notion of clock regions, thus imposing a strict relationship between the number of clock variables present in the models and the granularities that must be chosen in order to obtain the corresponding grid automaton. Also, instead of constructing a grid automaton directly, the notion of a region graph is first used to represent a possibly infinite transition system. Then, by a process of sampling, a grid automaton is derived and from this automaton test sequences can be extracted. Further, continuous time evolution is modeled by means of clock variables only, and context variables are not allowed.

Fouchal [22] proposes another method based on timed automata and test purposes, as sug-

gested in [11]. In Fouchal's proposal, region graphs are also used and are likewise sampled in order to obtain grid automata. Algorithms are used in an exhaustive test generation process, but no guarantees of correctness are offered. It is difficult to realize how precise are the timed test sequences that are obtained, specially when the original models are combined with test purposes. These models, also, do not deal with context variables.

In [23], Fouchal and co-workers present a test strategy similar to the one discussed here. Although both strategies are related, our work deals explicitly with dense time in order to capture timed properties, whereas in [23] a notion of timed elements is used to imitate continuous time evolution, in a process that offers no guarantees of accuracy of the test suites. Once again, context transformations and returning values to the environment are not considered.

Similar approaches appeared in [12, 17, 24], none of them dealing with context transformations. All of them use the classical notion of clock regions to obtain grid automata from which test case sequences can be extracted. But the large number of clock variables in typical models often lead to huge grids, due to the exponential number of clock regions and the need to enforce the strict relationship between the number of clocks and the chosen granularities. In contrast, our approach allows for an ample range of choices for appropriate granularity values, thus leading to more controllable grid automata and to more manageable test suites.

Petrenko and co-workers [13] offer a different approach. Their aim is to verify whether a test sequence yields the same outputs when applied to suspicious configurations. Suspicious configurations are obtained with the aid of expertise from system testers. Further, no treatment of continuous time evolution is provided on that proposal. In our work we can model both fail and desired system behavior, accommodating more general techniques for testing systems. Also, in contrast, we can handle timed systems with context variables.

Jeannet and co-workers [15] use the ioSTS formalism for specification models. In this work an enumeration approach of data values is used in order to avoid the state space explosion problem. But, since a discretization method is not used, it is problematic to capture continuous time evolution in an appropriate manner in these models, thus making it difficult to test timed properties. Moreover, the method can incur in high costs when calculating constraints using approximations in order to find test sequences. Further, the formal model and the proposed method do not deal simultaneously with timed requirements and context transformations.

7 Concluding Remarks

Methods and techniques for automatically generating test cases for critical and reactive systems have been proposed, many of which based on formal methods. Some deal with continuous time evolution, others allow some form of data flow. But a single formalism for treating both these issues simultaneously has been lacking. In this work we propose a method to automatically generate and apply test suites for timed systems with context variables.

The basic formal model here used is the Timed Input/Output Context Automaton (TIOCA), a generalization of the earlier Timed Input/Output Automaton (TIOA). Then, a general way of discretizing TIOCA was discussed and proven correct. This discretization technique avoids the classical notion of clock regions, and allows for an ample range of granularity values that could be chosen in the discretization process. The grid automaton thus obtained is capable of homomorphically simulating the original timed context system, and vice-versa. This lead to automatic method for generating test suites for systems that exhibit both continuous time evolution as well as context transformations. Full proofs of all claims can be found in [18].

In order to model specific system properties, we used the notion of test purpose models. Together with the notion of synchronous product, the discretization algorithm was able to generate grid automata that reflect both the behavior of the original timed context system, as well as properties specified by the test purpose. By automating the extraction of test cases from this grid automaton, the desired test suites were then constructed.

As for further studies along these lines, we can suggest a formal development of the notion of conformance testing. More efficient processes could be explored by considering the constructing of the grid automaton on-the-fly while extracting test sequences. The grid algorithm could also be used implicitly when testing runs. As another suggestion, shorter test suites might be obtained by factoring out common subwords from test cases.

References

[1] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, 2008, pp. 1–38.

[2] S. J. Cunning and J. W. Rozenblit, "Automating test generation for discrete event oriented embedded system s," *J. Intell. Robotics Syst.*, vol. 41, no. 2-3, pp. 87–112, 2005.

[3] B. Nielsen and A. Skou, "Test generation for time critical systems: Tool and case study," *ecrts*, vol. 00, p. 0155, 2001.

[4] J. Tretmans, "Testing concurrent systems: A formal approach," in *CONCUR '99: Proceedings of the 10th International Conference on Co ncurrency Theory*, ser. Lecture Notes in Computer Science, J. Baeten and S. Mauw, Eds., vol. 1664. London, UK: Springer-Verlag, 1999, pp. 46–65.

[5] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager, *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.

[6] R. Gawlick, R. Segala, J. F. Sogaard-Andersen, and N. A. Lynch, "Liveness in timed and untimed systems," in *Automata, Languages and Programming*, 1994, pp. 166–177.

[7] A. L. Bonifácio and A. V. Moura, "A New Timed Discretization Method for Automatic Test Generation for Timed Systems," Institute of Computing, University of Campinas, Tech. Rep. IC-09-31, September 2009.

[8] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[9] R. Alur, "Timed automata," in *CAV'99*, ser. LNCS, no. 1633, 1999.

[10] R. Cardell-Oliver, "Conformance tests for real-time systems with timed automata specifications," *Formal Aspects of Computing*, vol. 12, no. 5, pp. 350–371, 2000.

[11] A. En-Nouaary and R. Dssouli, "A guided method for testing timed input output automata," in *TestCom*, 2003, pp. 211–225.

[12] A. En-Nouaary, R. Dssouli, F. Khendek, and A. Elqortobi, "Timed test cases generation based on state characterization technique," in *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1998, p. 220.

[13] A. Petrenko, S. Boroday, and R. Groz, "Confirming configurations in efsm testing," *IEEE Trans. Softw. Eng.*, vol. 30, no. 1, pp. 29–42, 2004.

[14] C.-J. Wang and M. T. Liu, "Generating test cases for efsm with given fault models." in *INFOCOM*, 1993, pp. 774–781.

[15] B. Jeannet, T. Jéron, and V. Rusu, "Model-based test selection for infinite-state reactive systems," in *FMCO*, 2006, pp. 47–69.

[16] A. Petrenko and N. Yevtushenko, "Testing from partial deterministic fsm specifications," *IEEE Trans. Comput.*, vol. 54, no. 9, pp. 1154–1165, 2005.

[17] A. En-Nouaary, "A scalable method for testing real-time systems," *Soft. Quality Control*, vol. 16, no. 1, pp. 3–22, 2008.

[18] A. L. Bonifácio and A. V. Moura, "Generating test suites for timed systems with context variables," Institute of Computing, University of Campinas, Tech. Rep. IC-09-38, October 2009.

[19] K. G. Larsen and W. Yi, "Time abstracted bisimulation: Implicit specifications and decidability," *j-LECT-NOTES-COMP-SCI*, vol. 802, pp. 160–176, 1994.

[20] J. Tretmans, "Test generation with inputs, outputs, and quiescence." in *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, ser. Lecture Notes in Computer Science, T. Margaria and B. Steffen, Eds., vol. 1055. Springer, 1996, pp. 127–146.

[21] A. Gargantini, "Conformance testing," in *Model-Based Testing of Reactive Systems: Advanced Lectures*, ser. Lecture Notes in Computer Science, M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., vol. 3472. Springer-Verlag, 2005, pp. 87–111.

[22] H. Fouchal, "Conformance testing techniques for timed systems," in *SOFSEM*, 2002, pp. 1–19.

[23] H. Fouchal, E. Petitjean, and S. Salva, "Testing timed systems with timed purposes," in *RTCSA'00: Seventh International Conference on Real-Time Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2000, p. 166.

[24] A. En-Nouaary, R. Dssouli, and F. Khendek, "Timed wp-method: Testing real-time systems," *IEEE Trans. Softw. Eng.*, vol. 28, no. 11, pp. 1023–1038, 2002.

Epílogo

Um dos objetivos deste trabalho foi propor um modelo para capturar tanto a evolução contínua de tempo quanto as transformações de contexto de um sistema. O modelo formal, estendido dos TIOA, são autômatos temporizados com contexto denominados TIOCA.

Visando a geração de testes, usando o modelo TIOCA, se fez necessária a discretização de tais modelos. Essa proposta foi discutida e provada de maneira clara e precisa, de forma que a nova discretização pudesse obter autômatos grid mais compactos. Como consequência, casos de teste podem ser extraídos dos modelos discretizados, visto que os autômatos grid são capazes de simular de maneira homomórfica os modelos originais, e vice-versa.

Além da discretização, também usamos no trabalho a noção de proposta de teste para que propriedades específicas pudessem ser modeladas, diminuindo o número de testes a serem gerados. Por consequência dessa abordagem usamos um algoritmo para a obtenção do produto de TIOCA. A partir daí apresentamos uma estratégia de automação da extração de casos de teste usando o grid construído do produto entre a especificação e a proposta de teste.

Assim, o objetivo deste capítulo foi apresentar uma proposta para tornar o processo de geração de teste para sistemas complexos mais eficiente. O próximo capítulo apresenta algumas considerações finais, as conclusões da tese e possíveis direções de trabalhos futuros.

Capítulo 6

Conclusões

A derivação de casos de teste usando modelos formais permite a construção e verificação rígida de sistemas críticos de maneira confiável. Nota-se que existe uma relação muito próxima entre os tipos de sistemas que um dado modelo formal pode lidar e métodos disponíveis que derivam os conjuntos de testes. Daí o interesse em métodos eficientes de geração de casos de teste baseado em modelos para sistemas computacionais. Um dos modelos abordados nesta tese foram os tradicionais FSM, e algumas extensões, tais como as EFSM que englobam a noção de fluxo de dados através de variáveis de contexto.

Com o objetivo de gerar testes para um modelo mais complexo capaz de capturar não apenas o fluxo de dados, mas também a evolução contínua de tempo de sistemas de tempo real, foi proposto um novo modelo, denominado TEFSM, para lidar com ambos os aspectos. O método de derivação de sequências de confirmação proposto para TEFSM foi estendido de uma técnica usada em EFSM.

Porém, o modelo TEFSM possui entradas e saídas associadas, não permitindo que eventos externos influenciem as ações dos sistemas, gerando eventos observáveis para o meio externo. Além disso, a complexidade de se expressar propriedades de tempo e contexto num mesmo modelo, e também de aplicar uma abordagem de model-checking para a extração, nos levou a reconsiderar a estratégia inicial. Redirecionamos então o trabalho para, primeiramente, focar o problema de geração de casos de teste para as tradicionais FSM.

Nessa nova abordagem generalizamos o tradicional método W que gera conjuntos completos de teste para FSM. O método W tem como base o uso de conjuntos caracterização. Mostramos que o método W, na verdade, é um caso particular do nosso método G, e que também evita a computação de conjuntos caracterização, bem como dos índices das especificações. Destacamos que a demonstração dos resultados foi apresentada de forma clara, contendo provas detalhadas de corretude dos algoritmos. Como trabalhos futuros nesta direção, idéias similares poderiam ser usadas para estender e generalizar outros métodos de geração de casos de teste, tais como os métodos W_p e HSI.

Com a questão da geração de testes melhor fundamentada, usando um modelo formal mais simples, avançamos para um modelo mais complexo que captura o aspecto de tempo. Surgiu então a proposta de geração baseado no modelo TIOA, onde usamos uma nova técnica de discretização, desviando da discretização clássica através de regiões de relógio. Essa nova discretização possibilita uma diminuição considerável no número de estados do grid construído, já que usamos uma relação mais flexível na escolha da granularidade utilizada no processo de discretização. Mostramos que dado um modelo TIOA, este é capaz de simular o respectivo grid construído usando nosso método, e vice-versa. Este resultado serviu como base para a geração de casos de teste, visando propriedades específicas de um sistema. Através da noção de proposta de teste e do conceito de produto síncrono de TIOA foi possível utilizar uma estratégia de realimentação dos modelos para realizar o teste de conformidade. A proposta de teste é utilizada para especificar propriedades importantes de um sistema, enquanto o produto nos permite capturar o comportamento coordenado de uma especificação e uma propriedade modelada.

Um aprimoramento nesta linha seria o uso de uma estratégia *on-the-fly* na técnica de realimentação de subsequências. Outro ponto seria a utilização de uma representação simbólica das sequências de teste, resultando numa diminuição significativa das sequências temporizadas a serem manipuladas.

Com os resultados anteriores alcançados para o modelo TIOA, retomamos o objetivo principal da geração de testes para um modelo de tempo e contexto. Definimos então um novo modelo, chamado TIOCA, capaz de modelar aspectos de tempo e transformações de dados. Conseguimos estender a discretização do modelo TIOA, de forma a obter um grid para o modelo TIOCA. Provamos também que o modelo TIOCA é capaz de simular seu correspondente autômato grid, construído com essa proposta, e vice-versa. Com essa base sólida, estendemos os conceitos de geração de teste, proposto para TIOA, aplicados agora sobre o modelo TIOCA. Baseado na idéia de realimentação de sequências, mostramos como obter os vereditos de conformidade quando implementações candidatas são testadas.

Sugerimos como trabalhos futuros nessa linha, o desenvolvimento formal da noção de conformidade usando realimentação de sequências. Além disso, processos mais eficientes para explorar a construção de autômatos grid de modo *on-the-fly* e extrair sequências de teste, seriam de grande valia. Novamente, para essa generalização poderíamos também considerar a construção implícita das execuções de teste, usando estruturas simbólicas de representação. Isso permitiria a fatoração de subsequências comuns entre casos de teste gerados.

Referências Bibliográficas

[1] R. Alur. Timed automata. Em *CAV'99*, número 1633 in LNCS, 1999.

[2] Rajeev Alur e David Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Adilson Luiz Bonifácio e Arnaldo Vieira Moura. A New Timed Discretization Method for Automatic Test Generation for Timed Systems. Relatório Técnico IC-09-31, Institute of Computing, University of Campinas, September de 2009.

[4] Adilson Luiz Bonifácio e Arnaldo Vieira Moura. Generating test suites for timed systems with context variables. Relatório Técnico IC-09-38, Institute of Computing, University of Campinas, October de 2009.

[5] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, Adenilso da Silva Simão, e José Carlos Maldonado. Towards deriving test sequences by model checking. *Electron. Notes Theor. Comput. Sci.*, 195:21–40, 2008.

[6] A. L. Bonifácio, A. V. Moura, A. S. Simão, e J. C. Maldonado. Conformance Testing by Model Checking Timed Extended Finite State Machines. Em *Brazilian Symposium on Formal Methods (SBMF'06)*, pp. 43–58, Natal, setembro de 2006.

[7] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, e Adenilso da Silva Simão. A generalized model-based test generation method. Em Antonio Cerone e Stefan Gruner, editores, *Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM*, pp. 139–148, Cape Town, South Africa, 10–14, nov de 2008. IEEE Computer Society.

[8] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, e Adenilso da Silva Simão. Exponentially more succinct test suites. Relatório Técnico IC-09-07, Institute of Computing, University of Campinas, March de 2009. In English, 27 pages.

[9] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, e Adenilso da Silva Simão. A generalized model-based test generation method. Relatório Técnico IC-08-014, Instituto de Computação, Universidade Estadual de Campinas, Campinas, maio de 2008.

[10] Rachel Cardell-Oliver. Conformance tests for real-time systems with timed automata specifications. *Formal Aspects of Computing*, 12(5):350–371, 2000.

[11] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Transactions on Software Engineering*, 4(3):178–187, 1978.

[12] Steven J. Cunning e Jerzy W. Rozenblit. Automating test generation for discrete event oriented embedded system s. *J. Intell. Robotics Syst.*, 41(2-3):87–112, 2005.

[13] R. Dorofeeva, K. El-Fakih, e N. Yevtushenko. An improved conformance testing method. Em *Formal Techniques for Networked and Distributed Systems*, volume 3731 de *Lecture Notes in Computer Science*, pp. 204–218. Springer, 2005.

[14] A. En-Nouaary, R. Dssouli, F. Khendek, e A. Elqortobi. Timed test cases generation based on state characterization technique. Em *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, p. 220, Washington, DC, USA, 1998. IEEE Computer Society.

[15] Abdeslam En-Nouaary. A scalable method for testing real-time systems. *Soft. Quality Control*, 16(1):3–22, 2008.

[16] Abdeslam En-Nouaary e Rachida Dssouli. A guided method for testing timed input output automata. Em *TestCom*, pp. 211–225, 2003.

[17] Abdeslam En-Nouaary, Rachida Dssouli, e Ferhat Khendek. Timed wp-method: Testing real-time systems. *IEEE Trans. Softw. Eng.*, 28(11):1023–1038, 2002.

[18] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, e A. Ghedamsi. Test selection based on finite state models. *IEEE Transaction on Software Engineering*, 17(6), junho de 1991.

[19] A. Gargantini. Conformance testing. Em M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, e A. Pretschner, editores, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 de *Lecture Notes in Computer Science*, pp. 87–111. Springer-Verlag, 2005.

[20] R. Gawlick, R. Segala, J. F. Sogaard-Andersen, e N. A. Lynch. Liveness in timed and untimed systems. Em *Automata, Languages and Programming*, pp. 166–177, 1994.

[21] Mohammed Ghriga e Phyllis G. Frankl. Adaptive testing of non-deterministic communication protocols. Em *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pp. 347–362, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.

[22] G. Gonnenc. A method for the design of fault detection experiments. *IEEE Transactions on Computing*, 19:551–558, 1970.

[23] F. C. Hennie. Fault detecting experiments for sequential circuits. Em *FOCS*, pp. 95–110, 1964.

[24] R. M. Hierons. Separating sequence overlap for automated test sequence generation. *Automated Software Engg.*, 13(2):283–301, 2006.

[25] Bertrand Jeannet, Thierry Jéron, e Vlad Rusu. Model-based test selection for infinite-state reactive systems. Em *FMCO*, pp. 47–69, 2006.

[26] M. Krichen. State identification. Em M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, e A. Pretschner, editores, *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 de *Lecture Notes in Computer Science*, pp. 87–111. Springer-Verlag, 2005.

[27] M. Krichen e S. Tripakis. Black-box conformance testing for real-time systems. Em *Model Checking Software: 11th International SPIN Workshop*, número 2989 in Lecture Notes in Computer Science, pp. 109–126, Barcelona, Spain, abril de 2004.

[28] K. G. Larsen e W. Yi. Time abstracted bisimulation: Implicit specifications and decidability. *j-LECT-NOTES-COMP-SCI*, 802:160–176, 1994.

[29] Stephan Merz. Model checking: A tutorial overview. Em F. Cassez et al., editor, *Modeling and Verification of Parallel Processes*, volume 2067 de *Lecture Notes in Computer Science*, pp. 3–38. Springer-Verlag, Berlin, 2001.

[30] Brian Nielsen e Arne Skou. Test generation for time critical systems: Tool and case study. *ecrts*, 00:0155, 2001.

[31] Alexandre Petrenko, Sergiy Boroday, e Roland Groz. Confirming configurations in efsm testing. *IEEE Trans. Softw. Eng.*, 30(1):29–42, 2004.

[32] Alexandre Petrenko e Gregor v. Bochmann. Selecting test sequences for partially-specified nondeterministic finite state machines. Em Gang Luo, editor, *IWPTS '94: 7th IFIP WG 6.1 international workshop on Protocol test systems*, pp. 95–110, London, UK, UK, 1995. Chapman & Hall, Ltd.

[33] Alexandre Petrenko e Nina Yevtushenko. Testing from partial deterministic fsm specifications. *IEEE Trans. Comput.*, 54(9):1154–1165, 2005.

[34] Ali Rezaki e Hasan Ural. Construction of checking sequences based on characterization sets. *Computer Communications*, 18(12):911–920, 1995.

[35] D. Sidhu e T. Leung. Experience with test generation for real protocols. Em *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pp. 257–261, New York, NY, USA, 1988. ACM.

[36] Deepinder P. Sidhu e Ting kau Leung. Formal methods for protocol testing: A detailed study. *IEEE Trans. Softw. Eng.*, 15(4):413–426, 1989.

[37] Michael Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[38] Jan Tretmans. Test generation with inputs, outputs, and quiescence. Em Tiziana Margaria e Bernhard Steffen, editores, *Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS '96, Passau, Germany, March 27-29, 1996, Proceedings*, volume 1055 de *Lecture Notes in Computer Science*, pp. 127–146. Springer, 1996.

[39] Jan Tretmans. Testing concurrent systems: A formal approach. Em J.C.M Baeten e S. Mauw, editores, *CONCUR '99: Proceedings of the 10th International Conference on Co ncurrency Theory*, volume 1664 de *Lecture Notes in Computer Science*, pp. 46–65, London, UK, 1999. Springer-Verlag.

[40] Jan Tretmans. Model based testing with labelled transition systems. Em *Formal Methods and Testing*, pp. 1–38, 2008.

[41] Hasan Ural, Xiaolin Wu, e Fan Zhang. On minimizing the lengths of checking sequences. *IEEE Trans. Comput.*, 46(1):93–99, 1997.

[42] Chang-Jia Wang e Ming T. Liu. Generating test cases for efsm with given fault models. Em *INFOCOM*, pp. 774–781, 1993.