UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Gabriel Maicon Marcílio

Verificação Funcional Pós-Particionamento em Sistemas Integrados de Hardware e Software

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Luiz Cláudio Villar dos Santos, Dr.
(orientador)

Florianópolis, dezembro de 2008

Verificação Funcional Pós-Particionamento em Sistemas Integrados de Hardware e Software

Gabriel Maicon Marcílio

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na área de concentração "Sistemas de Computação" e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Frank Siqueira

Banca Examinadora

Luiz Cláudio Villar dos Santos, Dr. (orientador)

Flávio Rech Wagner, Ph.D.

Sandro Rigo, Dr.

Carlos Barros Montez, Dr.

The road goes ever on and on.
Down to the door where it began.
-Bilbo Baggins

À minha família

Agradecimentos

Ao professor Sandro Rigo e a Bruno Albertini que cederam a infra-estrutura de implementação utilizada neste trabalho.

Aos membros do LAPS e do LABSOFT por todo auxílio prestado.

À Minha mãe Albertina da Silva, meu irmão Marcos P. Marcílio e meu pai João P. Marcílio e a Karine M. Arasaki pela compreensão, estímulo, amor, paciência e incentivo.

Ao Professor Dr. Luiz Cláudio Vilar dos Santos pela orientação e incentivo que tornaram este trabalho possível.

Ao *Brasil Test Center* (BTC) da Motorola Industrial do Brasil pelo fomento parcial à produção deste trabalho, com bolsa paga no âmbito do projeto *Test Automation*.

À CAPES, no âmbito do Programa Nacional de Cooperação Acadêmica, pelo custeio parcial da execução deste trabalho (Processo nº 0326054).

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

DUV: Device Under Verification

EDA: Electronic Design Automation

ESL: Electronic System Level

IP: Intellectual Property

PBD Platform-Based Design

RTL: Register-Transfer Level

SoC: System-on-Chip

TTM: Time-to-Market

RGM: Reference Golden Model

SAT: Satisfability (Boolean satisfiability problem)

Lista de Símbolos

Monitor e seus espelhos:

m**:**	Um monitor
m^+**:**	Instância do monitor m no modelo de referência
m^-**:**	Instância do monitor m no DUV

Instâncias de módulo:

I**:**	Instância de um módulo de uma plataforma
I^+**:**	Representação do modelo de referência de um módulo I
I^-**:**	Representação do modelo sob verificação de um módulo I

Relações:

$u \approx v$**:**	Os valores u e v monitorados são compatíveis
R**:**	Relação de precedência entre eventos no RGM

Funções:

μ**:**	Um mapeamento próprio
$BVG(m)$**:**	Grafo bipartido de verificação associado ao monitor m
$Adj(v)$**:**	Conjunto de vértices adjacentes ao vértice v
$Adj(S)$**:**	Conjunto dos vértices adjacentes aos vértices do conjunto S

Elementos de Grafos:

V^+**:**	Conjunto de eventos monitorados no RGM
V^-**:**	Conjunto de eventos monitorados no DUV
E**:**	Conjunto de arestas de um grafo
M**:**	Um casamento arbitrário
$\mathcal{M}$**:**	Um casamento próprio para o BVG

C_j**:**	Componente conexo de um grafo bipartido
V_j**:**	Conjunto de vértices de um componente conexo
E_j**:**	Conjunto de arestas de um componente conexo
$\|Adj(v)\|$**:**	Grau do vértice v

Resumo

O escopo tradicional da verificação funcional foi ampliado com o surgimento dos fluxos de projeto em nível de sistema eletrônico (ESL). Nesses fluxos, logo após o particionamento hardware-software, a verificação precisa lidar com tipos abstratos de dados, com artefatos de implementação e com a possível não-preservação, no dispositivo sob verificação (DUV), da ordem dos comportamentos no modelo de referência (*golden model*). As técnicas existentes para verificação pós-particionamento estão limitadas pelo uso de heurísticas (que colocam em risco as garantias de verificação) ou por abordagens *black-box* (que restringem a observabilidade).

Este trabalho adota uma abordagem *white-box* e propõe uma nova técnica que opera sobre amostras de dados capturadas por monitores e armazenadas na forma dos assim-chamados *logs*. Para cada ponto a ser verificado, inserem-se monitores espelhados: um no modelo de referência, outro no DUV. A verificação automática dos *logs* é formulada com um problema de casamento (*matching*) em um grafo bipartido. O problema clássico foi modificado para capturar não apenas a compatibilidade de valores monitorados, mas também a precedência de eventos, de forma a viabilizar o tratamento da não-preservação da ordem no DUV.

A formulação adotada permitiu provar várias propriedades, as quais foram utilizadas como base teórica para determinar as garantias de verificação da técnica proposta. A implementação dos monitores utilizou infra-estrutura pré-existente baseada em reflexão computacional. São apresentados resultados experimentais que validam a formulação e os algoritmos propostos.

Abstract

The traditional scope of functional verification has been extended with the rise of electronic-system-level (ESL) design flows. In those flows, immediately after hardware-software partitioning, verification has to deal with abstract data, with implementation artifacts, and, possibly, with the non-preservation, by the device under verification (DUV), of the the order of behaviors at the golden model. Existing approaches are limited either by the use of greedy heuristics (jeopardizing verification guarantees) or by black-box approaches (impairing observability).

This work adopts a white-box approach and proposes a new technique that operates on data samples captured by monitors and stored in the form of so-called logs. For each point to be verified, mirrored monitors are inserted: one at the golden model, another at the DUV. The automatic verification of the logs of a pair of mirrored monitors is cast as a bipartite graph matching problem. The classical problem was modified to capture not only value compatibility, but also event precedence, so as to allow the treatment of the non-preserved event order at the DUV.

The adopted formulation allowed us to prove several properties, which were used as stepping stones for determining the verification guarantees of the proposed technique. The implementation of monitors relied on pre-existing infrastructure based upon computational reflection. Experimental results validate the formulation and the proposed algorithms.

Capítulo 1

Introdução

Depois de contextualizar este trabalho em relação aos paradigmas contemporâneos de projeto de sistemas integrados, este capítulo ilustra, através de dois exemplos, o problema abordado nesta dissertação.

Além de introduzir informalmente o problema-alvo e a abordagem proposta para a sua solução, os exemplos destacam dois dos principais efeitos resultantes da implementação, que precisam ser tratados por ferramentas de verificação:

- A existência de comportamentos implementados que não foram especificados (devido a artefatos de implementação);
- A não-preservação da ordem de comportamentos (devida à adoção de modelos atemporais em paradigmas de projeto contemporâneos e ao distinto tratamento de concorrência ao longo do fluxo de projeto).

1.1 O contexto histórico

A revolução dos sistemas integrados (SoCs: *Systems-on-Chip*) teve início em meados da década de 90, quando a tecnologia de semicondutores alcançou dimensões de cerca de 0,35 a 0,25 μm [MAR 03b]. Essa redução permitiu que os mais importantes elementos de um sistema computacional (processadores, memória e alguns periféricos) fossem integrados em uma mesma pastilha de silício (SoC).

Entre os anos de 1995 e 1999 a comunidade científica viveu um período de grande entusiasmo porque acreditava que, na virada do milênio, seria possível desenvolver SoCs num curto intervalo de tempo, atendendo assim às pressões do *time-to-market* (TTM).

Entretanto, as previsões feitas durante a primeira fase da revolução não se concretizaram. Não era simples integrar blocos de propriedade intelectual (IP: *Intelectual Property*) desenvolvidos por terceiros e, à medida que a complexidade dos sistemas crescia, a dificuldade de integrar (reusar) componentes aumentava. Isto ocorreu porque, no afã de promover o reuso, algumas áreas importantes foram negligenciadas (principalmente o desenvolvimento de novas metodologias e ferramentas de projeto) [MAR 03b].

Apesar disso, o mercado de IPs entre os grande fabricantes manteve seu crescimento entre os anos de 1993 e 2001 [MAR 03b]. Os projetos desta época foram desenvolvidos seguindo metodologias e padrões de projeto bem definidos.

Baseado nessas experiências preliminares, entre os anos de 2000 e 2002, surgiu a noção de projeto baseado em plataforma (PBD: *Platform-based design*) [SV 01].

1.2 O projeto baseado em plataforma

Pode-se definir esta abordagem de projeto como um método organizado para reduzir os tempos e o risco envolvidos no projeto e verificação de SoCs, através do intenso reuso de hardware e software. O PBD agrega grupos de componentes dentro de uma plataforma reutilizável [MAR 03b].

Em outras palavras, PBD é essencialmente uma combinação de uma arquitetura de referência com técnicas de reuso [BER 02].

Assim, cada instância de uma plataforma deriva de uma arquitetura de referência e é caracterizada pelos seus componentes programáveis. A utilização de componentes programáveis garante a flexibilidade necessária para oferecer suporte a um conjunto de aplicações que viabilizem a produção de uma plataforma [SV 01].

As plataformas CoreConnect (IBM) [IBM 08], TI OMAP^{TM} (Texas Intruments) [INS 08] e a Nexperia-DVP (Philips) [NXP 08] são exemplos conhecidos da utilização do paradigma de PBD.

1.3 O projeto em nível de sistema

A abordagem de projeto em nível de sistema eletrônico (ESL: *Electronic System Level*) é um conjunto de métodos que têm foco na utilização de níveis mais altos de abstração, visando aumentar a compreensão do sistema para viabilizar a implementação eficiente de um projeto [GM 07].

Embora o fluxo ESL possa ser idealizado como uma abordagem *top-down* (onde o projeto parte do nível mais alto, para o nível mais baixo de abstração) o reuso de componentes requer uma abordagem *meet-in-the-middle* (que envolve iterações *top-down* e *bottom-up*).

Segundo [GM 07], o fluxo de projeto usando a abordagem ESL pode ser dividido em seis etapas, ilustradas esquematicamente na Figura 1.1.

Na etapa de *especificação e modelagem* são identificadas as funcionalidades que o sistema deve atender, além de suas restrições. Geralmente, estas especificações são escritas utilizando linguagem natural.

A *análise pré-particionamento* consiste na escolha de modelos e algoritmos que serão utilizados na implementação do sistema. Esta escolha deve levar em conta aspectos como desempenho, área ocupada em silício, potência consumida e complexidade.

A etapa de *particionamento* consiste em definir quais algoritmos escolhidos nas etapas anteriores devem ser implementados em software e quais em hardware.

Na etapa de *análise pós-particionamento* os modelos concebidos durante a análise pré-particionamento são refinados para refletir o particionamento adotado. A partir desta representação do sistema já é possível efetuar algumas estimativas de desempenho, potência e custo [GM 07]. É possível que ocorram algumas

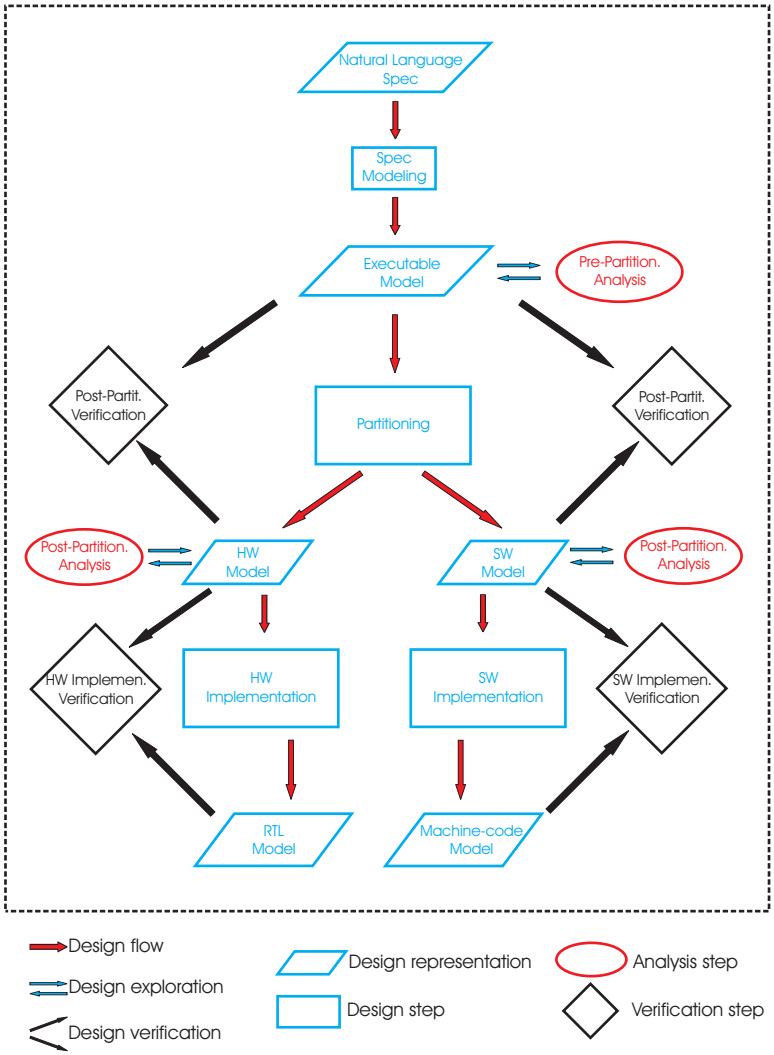

Figura 1.1: O fluxo de projeto usando a abordagem ESL

iterações entre a etapa de análise e particionamento visando alguma otimização no sistema. Este processo é conhecido como exploração do espaço de projeto (*design space exploration*).

Durante a *verificação pós-particionamento*, a equipe de projeto deve avaliar se a descrição do sistema, obtida após o particionamento, preservou o comportamento especificado. Essa avaliação ocorre através da comparação do modelo de referência com o modelo resultante do particionamento.

A verificação inicia-se a com a elaboração de um plano de verificação

(*verification plan*). Neste documento, fica registrado o escopo da verificação, as entidades que deverão ser monitoradas e os testes que devem ser realizados para validar o sistema. Assim, esse plano é utilizado como base para a implementação do ambiente de verificação.

A *implementação do hardware* consiste na criação de modelos que, futuramente, poderão ser implementados através de síntese comportamental, lógica e física. Algumas decisões de projeto que impactam diretamente o desempenho do sistema devem ser tomadas nesta etapa, tais como o como compartilhamento de recursos e as inserções de estruturas de *pipeline*.

Tradicionalmente, a *implementação do software* de um sistema era iniciada quando uma versão estável do hardware já estava disponível. Numa abordagem ESL, o software e o hardware são desenvolvidos paralelamente, para acelerar o desenvolvimento do projeto (*hardware-software codesign*).

Os refinamentos de hardware e software produzidos durante esta etapa devem ser submetidos ao ambiente de verificação elaborado na etapa anterior, ou seja, faz-se a *verificação da implementação*. Esta interação entre a etapa de implementação e a etapa de verificação assegura a compatibilidade entre a diferentes representações do modelo implementado.

O tópico de pesquisa abordado nesta dissertação é a verificação funcional de *representações executáveis* de blocos de hardware e software (que podem ser obtidas utilizando, por exemplo, SystemC [OSC 08]). Embora o foco de aplicação da técnica aqui proposta resida na verificação pós-particionamento (para a qual há um menor número de técnicas disponíveis), isso não exclui sua aplicação para a verificação da implementação, desde que representações executáveis estejam disponíveis para isso.

1.4 A verificação de sistemas

A história da área de verificação e o seu papel crucial na automação de projeto eletrônico (EDA: *Eletronic Design Automation*) resultou num grande

número de técnicas e ferramentas para verificar se uma dada implementação está de acordo com uma especificação. Livros inteiros já foram escritos sobre este importante assunto [MAR 03a, BW 03, SI 04, BAI 05, MEY 04], discutindo aspectos importantes da verificação de um sistema como a geração correta de vetores de estímulo, a cobertura de testes, as asserções temporais, os ambientes e as linguagens de verificação.

Com o surgimento da abordagem ESL, o escopo da verificação foi ampliado. Além da tradicional verificação da implementação, surgiu a noção de verificação pós-particionamento.

Neste escopo estendido, a verificação tem que lidar com resoluções distintas [BB 05] para dados (*token*, propriedade, valor, formato, sinal lógico) e para temporização (eventos parcialmente ordenados, eventos do sistema, ciclo de instrução, precisão de ciclos e precisão de tempo). Por exemplo, valores de dados a serem monitorados podem estar associados a atributos de uma representação orientada a objeto de um módulo de um sistema (eles não estão mais limitados a saídas *bit-true* de um componente do hardware).

Além disso, a verificação deve levar em conta a utilização de diferentes níveis de abstração ao longo de um projeto. Assim, estabelecer a compatibilidade entre modelos tem um papel crucial no projeto de um sistema. Ou seja, deve ser possível estabelecer uma correspondência entre duas representações em diferentes níveis de abstração de mesmo projeto quando submetidos a uma mesma seqüência de estímulos. Para estabelecê-la, costuma-se utilizar abordagens com diferentes graus de observabilidade. Abordagens do tipo *black-box* têm acesso somente a elementos externos (entradas e saídas), enquanto abordagens do tipo *white-box* possuem uma observabilidade maior, permitindo acesso a elementos internos (atributos).

Um aspecto crucial é como tornar observáveis os atributos de um objeto, para permitir a comparação entre implementação e comportamento especificado. Outro aspecto importante é como analisar automaticamente os registros históricos de eventos monitorados, que chamaremos de *logs*.

Um aspecto fundamental da verificação pós-particionamento diz

respeito à não-preservação da ordem de comportamentos no modelo sob verificação (DUV: *Device Under Verification*), como será discutido na Seção 1.6.

1.5 O problema-alvo e a abordagem proposta

Assuma que um módulo capaz de realizar operações de multiplicação é o módulo sob verificação (DUV: *Device Under Verification*). A Figura 1.2 mostra um ambiente de verificação típico. Um modelo de referência (RGM: *Reference Golden Model*) representa os comportamentos especificados. Para cada estímulo gerado (`multiplicand` e `multiplier`) um comparador verifica se a saída obtida equivale à saída esperada (`product`). Isto resulta numa abordagem *black-box*.

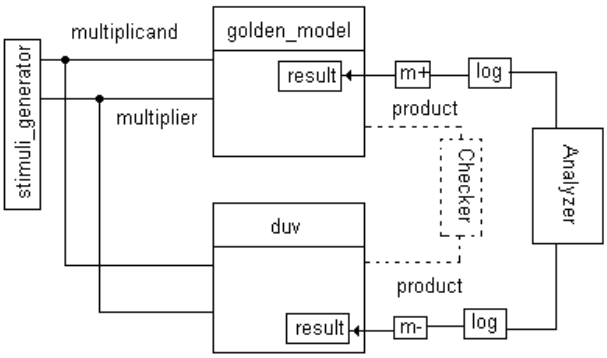

Figura 1.2: Um exemplo de ambiente de verificação

Entretanto, suponha que um engenheiro de verificação precisa inspecionar elementos internos ao DUV (por exemplo, uma variável). A técnica proposta neste trabalho viabiliza este enfoque *white-box*, anexando um *monitor* a um atributo, variável, sinal ou registrador de um DUV.

Na Figura 1.2, duas instâncias de um monitor m (m^+ e m^-) foram anexadas a uma variável interna (`result`). O histórico dos valores observados é armazenado na forma de um *log*.

A Figura 1.3 mostra uma representação SystemC [OSC 08] para o

modelo de referência. Note que ela implementa o comportamento desejado, empregando diretamente uma operação de multiplicação para calcular o produto (linha 17).

```
1  SC_MODULE(golden_model){
2      ...
3      sc_in<int> multiplier;
4      sc_in<int> multiplicand;
5      sc_out<int> product;
6
7      int result; //Monitor m+ will be attached to this variable
8      void run()
9      {
10       while(true)
11       {
12               wait();
13               result = 0;
14               int factor1 = multiplier->read();
15               int factor2 = multiplicand->read();
16
17               result = factor1 * factor2;
18
19               product->write(result);
20       }
21
22     }
23    ...
24  }
```

Figura 1.3: Uma representação para o modelo de referência

Assuma que, como resultado de um refinamento visando uma implementação mais barata, o módulo na Figura 1.4 foi obtido. Note que as operações de soma utilizadas para calcular o produto (linha 19) podem ser consideradas como efeitos colaterais da implementação da funcionalidade desejada (multiplicação). Estes efeitos colaterais são chamados de *artefatos de implementação* [GM 07].

Devido ao uso de um artefato de implementação, os valores monitorados em m^- vão apresentar comportamentos não especificados, os quais devem ser identificados e separados daqueles especificados no modelo de referência.

A Figura 1.5 mostra a descrição do gerador de estímulos, que foi preparado para disparar duas operações sucessivas de multiplicação.

Na técnica aqui proposta, toda mudança de valor observada por

```
1  SC_MODULE(duv){
2      ...
3      sc_in<int> multiplier;
4      sc_in<int> multiplicand;
5      sc_out<int> product;
6
7      int result; //Monitor m- will be attached to this variable
8      void run()
9      {
10       while(true)
11       {
12               wait();
13               int factor1 = multiplier->read();
14               int factor2 = multiplicand->read();
15               result = 0;
16
17               int i;
18               for(i=0; i<factor1; i++)
19                    result += factor2;
20
21               product->write(result);
22       }
23
24     }
25    ...
26 }
```

Figura 1.4: Uma representação para o DUV

uma instância de um monitor é representada como uma entrada num *log*.

Um *log* começa quando um valor inicial é atribuído a um elemento e captura cada novo valor observado durante a seqüência de eventos disparada pelos estímulos aplicados.

Para o padrão de estímulos gerados, os *logs* das instâncias m^+ e m^- são $\langle 0, 12, 0, 6\rangle$ e $\langle 0, 3, 6, 9, 12, 0, 3, 6\rangle$, respectivamente. Para o caso particular deste exemplo, vamos convencionar que dois valores só são compatíveis se forem idênticos.

Para verificar se um comportamento especificado no modelo de referência está implementado no DUV, vamos estabelecer uma correspondência entre valores compatíveis nos logs de m^+ and m^-. Para analisar esta correspondência, vamos construir um grafo bipartido, onde cada partição, digamos V^+ (V^-), representa o log de um monitor m^+ (m^-). Uma aresta (v^+, v^-) significa que os valores associados aos vértices v^+ e v^- são compatíveis.

A Figura 1.6 mostra o grafo bipartido resultante, onde o valor asso-

```
1  SC_MODULE(stimuli_generator){
2      ...
3      sc_out<int> multiplier;
4      sc_out<int> multiplicand;
5
6      void run()
7      {
8                  wait();
9                  multiplier->write(4);
10                 multiplicand->write(3);
11
12                 wait();
13                 multiplier->write(2);
14                 multiplicand->write(3);
15
16                 wait();
17                 sc_stop();
18     }
19    ...
20 }
```

Figura 1.5: Uma representação para o gerador de estímulos

ciado a um vértice está nele rotulado. Este grafo será denominado de *grafo bipartido de verificação* (BVG).

Um casamento (*matching*) em um grafo bipartido é um subconjunto de suas arestas tais que duas arestas não compartilhem um vértice. Desta forma, se encontrarmos um casamento M tal que cada vértice pertencente a V^+ esteja casado, podemos concluir que cada comportamento especificado está implementado pelo DUV. Assim, os comportamentos especificados podem ser distinguidos dos não especificados.

Entretanto, nem todo casamento é apropriado. Como as mudanças de valores nos monitores são provocadas por uma seqüência de eventos, para preservar a *causalidade*, nenhuma resposta deverá preceder o evento que a disparou.

Por exemplo, o evento v_2^+ representa a inicialização de `result` (linha 13 na Figura 1.3) para o segundo estímulo aplicado (linhas 13-14 Figura 1.5), enquanto o evento v_3^+ representa a atribuição do produto a `result` (linha 17 na Figura 1.3) para o segundo estímulo aplicado (linhas 13-14 a Figura 1.5). Assim, o evento v_2^+ precede o evento v_3^+ na especificação. Se escolhêssemos (v_2^+, v_5^-) para

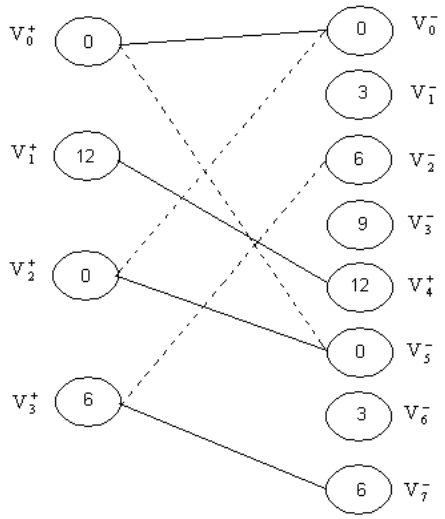

Figura 1.6: Um grafo bipartido e um casamento próprio

estar no casamento, então (v_3^+, v_2^-) não poderia nele estar, já que a atribuição do produto final a `result` (representada por v_2^-) no DUV precederia sua inicialização (representada por v_5^- no DUV). Este comportamento não-causal com certeza não reflete o comportamento desejado e deve ser excluído.

Para manter a causalidade, precisamos levar em conta que os conjuntos V^+ e V^- estão ordenados pela seqüência de eventos especificados, isto é, $V^+ = (v_0^+, v_1^+, v_2^+, v_3^+)$ e $V^- = (v_0^-, v_1^-, v_2^-, v_3^-, v_4^-, v_5^-, v_6^-, v_7^-)$

Na Figura 1.6, o conjunto de arestas destacadas representa um casamento que preserva a causalidade e, simultaneamente, garante que cada vértice em V^+ está casado. Um casamento com essas propriedades será denominado de *casamento próprio*. Desta forma, ele indica que todos os comportamentos especificados foram implementados. Se não for possível obter um casamento com estas características, uma travessia no grafo vai apontar qual evento especificado causou a anomalia. Isto torna a técnica aqui proposta útil para encontrar defeitos.

Por simplicidade, o exemplo assume que o DUV preserva a ordem dos eventos especificados no modelo de referência. Entretanto, há situações reais em que a ordem não é preservada pelo DUV para todos os eventos, como será discutido

na próxima seção.

1.6 O efeito da não-preservação da ordem de comportamentos

Um dos primeiros passos em um fluxo de projeto ESL é a criação de uma representação comportamental (funcional) que obedeça à especificação, dando origem a um modelo da especificação (declarativo ou executável). Entretanto, por razões de eficiência, esse modelo deve abstrair aspectos de temporização, pois a maior complexidade computacional de modelos temporizados tornaria proibitivo o co-projeto de hardware e software [GHE 05]. Além disso, o uso de uma representação executável temporizada tornaria não-reusáveis os *testbenches* [GM 07], pois teriam de ser retrabalhados toda vez que houvesse uma mudança no projeto impactando a temporização.

A necessidade de um modelo comportamental atemporal leva, entretanto, a duas conseqüências para a verificação, a seguir discutidas e ilustradas através de exemplos:

Consequência 1: A verificação pós-particionamento deve checar a ordem dos eventos, mas não sua temporização.

Suponha que uma especificação dite o seguinte requisito: um evento *Grant* ocorre sempre após um evento *Request* dentro de no máximo 4 ciclos. Quando a especificação é formalizada em um modelo atemporal (Modelo 1), apenas a ordem dos eventos é capturada: o evento *Grant* sucede o evento *Request*. A especificação poderia ter sido capturada em um modelo temporal aproximado (Modelo 2) da seguinte forma: o evento *Grant* ocorre sempre 2 ciclos após o evento *Request* (o que satisfaz o requisito original). Entretanto, se após a implementação, o evento *Grant* ocorre sempre 3 ciclos após o evento *Request*, ele satisfaz o Modelo 1, mas não o Modelo 2, embora satisfaça a especificação. Por isso, o modelo comportamental não deveria capturar a temporização em si, mas apenas a relação de precedência de even-

tos, que é condição necessária, mas não suficiente, para a validade da temporização.

Conseqüência 2: A ordem de eventos pode não ser preservada no DUV devido ao tratamento de concorrência adotado no modelo comportamental.

Suponha que uma especificação não imponha qualquer ordem entre os eventos *Request 1* e *Request 2* nem entre os eventos *Grant 1* e *Grant 2*, embora especifique a seguinte relação de precedência (*Request 1*, *Grant 1*) e (*Request 2*, *Grant 2*). Assuma que a especificação é formalizada em um modelo atemporal a ser usado como referência para verificação (RGM). Suponha agora que, durante a execução deste modelo, os eventos *Request 1* e *Request 2* ocorram simultaneamente. Suponha ainda que a forma como o RGM trate a concorrência dos eventos resulte em disparar o evento *Grant 1* antes do evento *Grant 2*, ou seja (*Grant 1*, *Grant 2*). Finalmente, assuma que o modelo do DUV receba *Request 2* antes de *Request 1* devido a diferentes atrasos de propagação decorrentes da implementação. Neste cenário, a ordem dos eventos no DUV seria (*Grant 2*, *Grant 1*), ao contrário da ordem induzida no modelo atemporal de referência (RGM). Assim, para o mesmo conjunto de estímulos de entrada, os comportamentos de saída observados entre RGM e DUV não exibiriam a mesma ordem de eventos. O que parece ser um indício de erro de implementação é na verdade conseqüência da ausência de uma restrição na especificação.

A não-preservação da ordem dos eventos pelo DUV é talvez um dos aspectos mais complexos da verificação em um fluxo de projeto ESL [GM 07].

A próxima seção ilustra, através de um exemplo propositadamente simples, como esta dissertação aborda estes aspectos.

1.7 Um exemplo ilustrativo

A Figura 1.7a ilustra uma especificação em que não há restrição alguma para a ordem relativa entre as escritas de y e z, mas apenas estabelece que a escrita de x deve precedê-las.

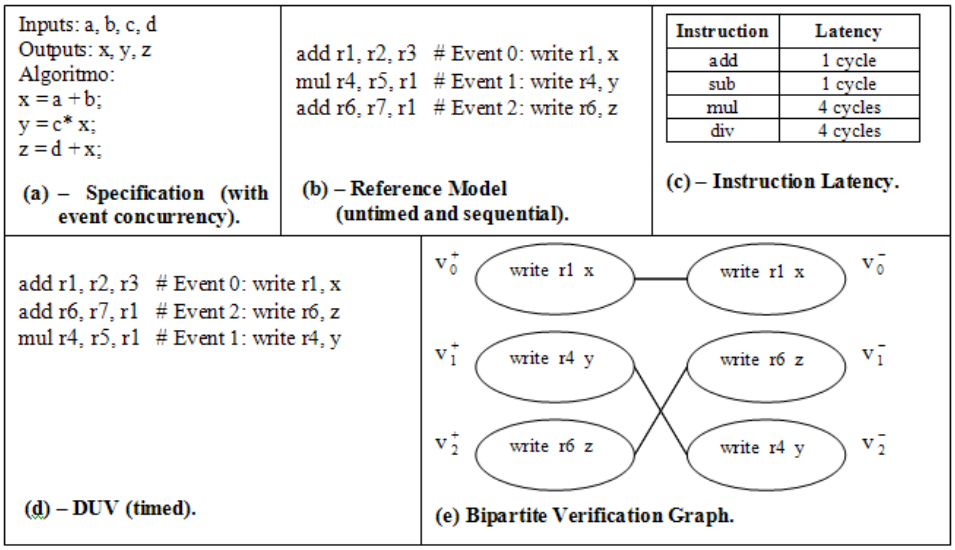

Figura 1.7: Exemplo de execução fora de ordem

Suponha que os comportamentos especificados venham a ser implementados em um co-processador aritmético. Assuma que se adote um modelo atemporal de referência para capturar a especificação, o qual trate a concorrência dos eventos através de sua mera serialização, na ordem em que as instruções são buscadas. Ora, este seria exatamente o caso típico em que um simulador do conjunto de instruções (puramente funcional) fosse utilizado para a modelagem. A Figura 1.7b ilustra esquematicamente o efeito desse modelo atemporal, que assume execução seqüencial. Além disso, ela mostra a correspondência entre as instruções no modelo de referência e os respectivos eventos de escrita no banco de registradores. Um evento de escrita, denotado por *write* r, v, representa a escrita do valor v no registrador-destino r.

Considere agora que, para a implementação do co-processador, seja adotada uma micro-arquitetura que suporte o bem-conhecido algoritmo de Tomasulo [PAT 07], para explorar a concorrência de operações independentes em unidades funcionais distintas. Assuma que as latências das instruções sejam aquelas representadas na Figura 1.7c. Suponha que existam estações de reserva distintas e livres para operações de soma e de multiplicação, conforme ilustra a Figura 1.8. Suponha

também que um monitor seja adicionado à porta de escrita do banco de registradores.

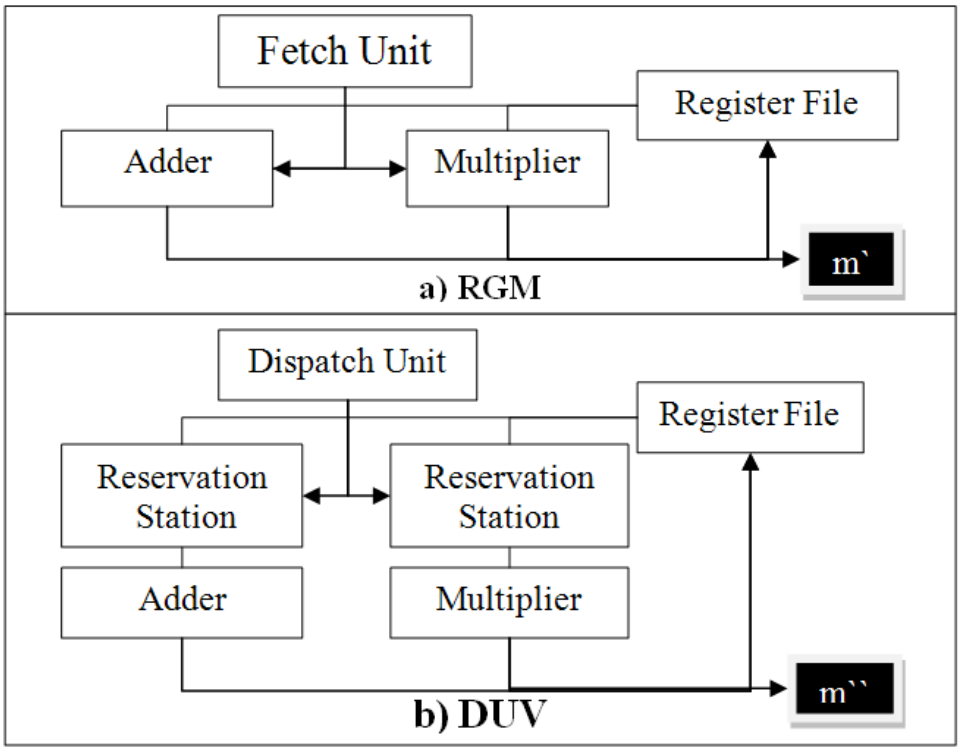

Figura 1.8: Representação do co-processador hipotético

A Figura 1.7d exibe a ordem efetiva de execução das intruções quando submetidas ao DUV. Nessa implementação, o Evento 1 ocorrerá ao final do primeiro ciclo, quando as demais instruções serão simultaneamente iniciadas. Devido às diferentes latências, o Evento 3 ocorrerá ao final do segundo ciclo; o Evento 2, ao final do quinto ciclo.

A Figura 1.7e ilustra o grafo bipartido de verificação e o casamento que corresponde à compatibilidade dos comportamentos especificados. Note que o cruzamento entre as arestas (v_1^+, v_2^-) e (v_2^+, v_1^-) não pode ter a interpretação de erro de implementação, pois a ordem dos eventos resultou invertida no DUV devido ao distinto tratamento de concorrência entre modelos.

Seja R $=\{(v_0^+, v_1^+), (v_0^+, v_2^+)\}$ a relação que captura a seguinte precedência entre eventos: (Evento 0, Evento 1) e (Evento 0, Evento 2).

Note que para decidir se o casamento obtido no BVG da Figura 1.7(e) satisfaz o princípio da causalidade, teve-se que analisar se a troca de ordem dos eventos no DUV satisfaz a relação de precedência R.

Portanto, não é possível analisar a compatibilidade de comportamentos sem uma relação de precedência de eventos que permita distinguir se a ordem induzida pelo DUV viola ou não a especificação. Note que essa relação é inerente à própria especificação, a partir da qual pode ser extraída, e não uma restrição imposta à resolução do problema.

No caso específico da abordagem aqui proposta, essa relação de precedência será a chave para distinguir casamentos próprios de impróprios. Como os algoritmos clássicos de casamento, ao procurar um casamento de maior cardinalidade, não assumem qualquer relação adicional entre seus vértices, exceto pela relação de compatibilidade capturada pelas arestas do grafo bipartido, tais algoritmos não podem ser diretamente reusados para resolver o problema da verificação que será formulado na Seção 3.2. Em outras palavras, o simples reuso de algoritmos clássicos não é adequado porque o casamento resultante poderia não obedecer ao princípio da causalidade.

Por isso, um novo algoritmo precisa ser concebido para restringir as soluções que seriam obtidas por algoritmos clássicos, instrumentando-os para distingüirem entre arestas pertencentes ou não a um casamento próprio.

1.8 O escopo e a contribuição desta dissertação

As premissas da abordagem adotadas nesta dissertação são as seguintes:

Premissa 1: A técnica proposta assume que a geração de estímulos para um dado componente foi tratada no ambiente de verificação de forma a garantir adequada cobertura.

Premissa 2: A técnica proposta aborda verificação de resposta aos estímulos e diagnóstico de erros baseada apenas no modelo de referência (verificadores temporais e de dados estão fora do escopo desta dissertação).

Premissa 3: A análise proposta é inerentemente limitada ao universo de comportamentos amostrados pelos monitores inseridos.

Premissa 4: A técnica proposta assume que se um monitor amostra no DUV valores não especificados no RGM, os comportamentos não especificados referem-se a artefatos de implementação.

Assim, se um dado comportamento especificado no modelo de referência fosse duplicado no DUV (por exemplo, se um *request* gerasse dois *grants*), a anomalia não seria capturada pela técnica proposta).

Premissa 5: A técnica proposta supõe que os elementos monitorados no RGM sejam preservados no DUV e que, portanto, faça sentido introduzir monitores no mesmo elemento nas duas representações.

Desta forma, se o refinamento de um modelo não preservasse algum elemento monitorado no modelo de alto nível, deveria ser criada alguma relação de mapeamento para viabilizar indiretamente a amostragem do elemento desejado.

Premissa 6: A técnica proposta pressupõe que se possa extrair da especificação informações sobre causa e efeito de eventos, que serão capturadas por uma relação de precedência R.

A relação R é dependente do domínio de aplicação. Por exemplo, ela pode capturar as dependências de dados e de controle de um código embarcado, a ordem de sinalização em um protocolo *handshaking*, etc.

As principais contribuições desta dissertação são:

- A viabilização do tratamentos do efeito de não-preservação da ordem de comportamentos sem limitar a observabilidade, no âmbito de uma abordagem *white-box*.

- O estabelecimento de garantias formais para a qualidade da verificação, como resultado da formulação escolhida para o problema, amparando-se em propriedades de casamento em grafos bipartidos.

Não é do conhecimento do autor a existência de técnicas que tratem a não-preservação de ordem em abordagem *white-box*, nem a existência de trabalhos correlatos que se amparem em propriedades de grafos bipartidos para fornecer garantias de qualidade da verificação.

1.9 A organização desta dissertação

Esta dissertação está organizada da seguinte maneira: O Capítulo 2 apresenta inicialmente uma breve revisão dos trabalhos correlatos nas áreas de verificação dinâmica, verificação formal e verificação em plataformas e concentra-se depois na análise de abordagens convencionais para o tratamento da não-preservação da ordem de comportamentos. O Capítulo 3 inicialmente formaliza algumas noções básicas que são em seguida utilizadas para formular o problema-alvo; ao final, provam-se algumas propriedades fundamentais a serem preservadas durante a verificação. O Capítulo 4 propõe algoritmos para resolver o problema-alvo e analisa as garantias de verificação, apresentando provas da ausência de falso negativo, além de discutir as condições de ocorrência de falsos positivos. O Capítulo 5 discute a implementação e apresenta resultados experimentais como evidência da correção da abordagem. O Capítulo 6 resume as principais conclusões e indica as perspectivas de trabalhos futuros.

Capítulo 2

Trabalhos correlatos

Como a literatura sobre verificação funcional é muito vasta, neste capítulo serão analisados algumas técnicas recentes mais próximas do foco deste trabalho, no âmbito das abordagens ESL e PBD. A primeira seção aborda algumas técnicas de verificação funcional enquanto a segunda seção aborda técnicas utilizadas para tratar a não-preservação da ordem de comportamentos.

2.1 Principais abordagens para verificação funcional

2.1.1 Verificação formal

A verificação formal de um sistema consiste na utilização de dispositivos formais ou matemáticos para provar que o sistema avaliado está correto. Uma das formas utilizadas consiste em estabelecer uma equivalência entre modelos em diferentes níveis de abstração. Se for possível provar que um modelo de alto nível, previamente validado, é equivalente a um modelo de baixo nível, então considera-se este último como validado.

Esta validação do modelo de baixo nível em função da validação prévia de um modelo em maior nível de abstração acelera o desenvolvimento do

sistema, já que a simulação em modelos em níveis mais baixos de abstração é muito mais lenta do que a simulação em modelos em mais alto nível.

Uma das técnicas utilizadas para estabelecer a compatibilidade entre modelos é o uso de pontos de corte (*cutpoints*). A noção de *cutpoints* está baseada na suposição de que se dois circuitos são funcionalmente similares, então deve ser possível definir um conjunto de pontos logicamente equivalentes, os quais não requerem verificação no modelo de baixo nível.

O trabalho de Feng e Hu [FEN 97] utilizou a noção de *cutpoints* para estabelecer equivalência entre um modelo SystemC (com precisão de pinos e de ciclos) e um modelo RTL. Para estabelecer esta equivalência, os autores comparam representações funcionais (obtidas através de simulação simbólica) das implementações em diferentes níveis de abstração. Esta comparação permite determinar pontos logicamente equivalentes, diminuindo o esforço total de verificação do sistema.

Outro ramo da verificação formal utiliza asserções para validar um modelo. O trabalho de Kasuya [KAS 07] propõe o uso das assim-chamadas asserções nativas em SystemC (NSCa: *Native SystemC assertion*) para modelar asserções com precisão de ciclos em SystemC. Por um lado, esta técnica viabiliza o reuso de asserções nos diferentes níveis de abstração, o que reduz o esforço de verificação. Por outro lado, ela é intrusiva, pois requer mudanças no DUV para inserção das asserções.

Em [HAB 06] asserções são utilizas para avaliar a cobertura funcional de um dado conjunto de vetores de verificação. Para avaliar a cobertura, os autores utilizam um modelo de referência (RGM) escrito na linguagem denominada AsmL (*Abstract State Machine Language*). Asserções são inseridas nas variáveis de estado do modelo de referência e avaliadas sempre que o valor da variável correspondente tem o seu valor alterado, ou seja, o modelo tem o seu estado alterado. Assim, a cobertura de um dado vetor v_i é avaliada pela razão entre o número de estados assumidos pelo modelo quando estimulado por v_i e número de estados possíveis. Após a avaliação da cobertura de um determinado conjunto de vetores, os autores sugerem o uso de algoritmos genéticos para aumentar a cobertura funcional através

de otimização da geração de vetores de verificação. Entretanto, o uso desta técnica requer que o engenheiro de verificação defina quais são as variáveis de estado de um DUV (onde estão inseridas as asserções). Ademais, o uso de algoritmos genéticos requer instrumentação para capturar informações específicas do domínio de aplicação: qual a representação de cromossomos através de uma codificação dos estímulos, além das operações de herança, mutação e recombinação para a codificação adotada.

Em [COR 00] propõe-se uma representação baseada em redes de Petri para a verificação de um sistema, a qual é denominada PRES+ (*Petri net based Representation for Embedded Systems*). Nesta representação as marcações nos lugares da rede representam os estados do sistema, enquanto as transições representam as operações realizadas pelo sistema. Cada transição possui anotações tais como, por exemplo, as condições necessárias para o seu disparo, além de informações como o tempo máximo necessário para realizar a transição. Através da representação PRES+, é possível avaliar-se, por exemplo, para que entradas um determinado estado do sistema é alcançado, ou qual o intervalo mínimo de tempo necessário para alcançar um determinado estado. No entanto, segundo [VAR 07] representações que enumerem todos os estados possíveis estão limitadas pelo crescimento exponencial do número de estados, inviabilizando assim a simulação de modelos mais complexos.

Como nem sempre é possível provar que dois modelos são equivalentes, Vardi [VAR 07] analisou algumas técnicas formais para estabelecer sua compatibilidade. Nesta abordagem, um modelo beneficia-se da verificação previamente realizada em modelo de alto nível compatível, permitindo que a verificação de cada implementação foque nos aspectos relevantes para aquele modelo de abstração. Por exemplo, a verificação de um modelo RTL pode se limitar aos artefatos de implementação inseridos, enquanto a verificação nos níveis superiores esta focada nas estruturas (particionamento) e algoritmos escolhidos.

2.1.2 Verificação dinâmica

A verificação dinâmica está associada à simulação do sistema. Tradicionalmente um ambiente de verificação é composto de um gerador de estímulos, um módulo a ser verificado (DUV) e um monitor. A função do gerador de estímulos é a de aplicar um conjunto de estímulos no DUV. Já o monitor deve capturar as saídas do DUV e verificar se seus valores estão corretos.

Em níveis mais baixos de abstração, a grande quantidade de detalhes torna proibitiva a simulação de todo o sistema. Além disso, em alguns casos, o ambiente de verificação precisa ser adaptado para incluir informações de baixo nível (por exemplo, o ambiente precisa ser alterado para capturar detalhes do protocolo de comunicação do DUV).

Uma alternativa para resolver o problema de reduzir o tempo de simulação do sistema é a co-simulação de componentes em diferentes níveis de abstração [SWA 06]. Assim, embora o DUV seja descrito em uma linguagem de descrição de hardware (como VHDL ou Verilog), os demais componentes podem ser descritos em níveis mais altos de abstração (C ou C++).

Para reaproveitar o ambiente de verificação, uma alternativa é a tradução automática de estímulos [BOM 06]. Esta abordagem utiliza *transactors*, que são um componentes responsáveis pela tradução de mensagens entre o ambiente de verificação (que utiliza chamadas de alto nível, como *write* ou *read*) e o DUV (que pode utilizar, por exemplo, sinais lógicos). Desta forma, o *transactor* reduz consideravelmente o esforço de implementação do ambiente de verificação. Outra vantagem é que, se o protocolo de comunicação do DUV for alterado, apenas o *transactor* deverá ser alterado para refletir a mudança (sem a utilização de um *transactor*, poderia ser necessário re-implementar todo o conjunto de rotinas de verificação).

No entanto, as técnicas citadas não são suficientes para verificar sistemas muito complexos. Para isto as plataformas costumam contar com mecanismos próprios adicionais, como será mostrado na próxima seção.

2.1.3 Suporte à verificação em plataformas comerciais

O reuso de componentes promovido pela abordagem PBD diminui o esforço total de verificação de um sistema, pois os módulos que compõem uma plataforma já foram verificados previamente. Desta forma, a verificação deve estar focada na interação entre os componentes.

Para sua plataforma *CoreConnect*, a IBM desenvolveu o TOS (Test Operation System)[MG 03], que é constituído de um *kernel*, uma API , um padrão e um ambiente de desenvolvimento. O *kernel* do TOS é desenvolvido em C e fornece algumas facilidades para a execução de rotinas de verificação, tais como a inicialização dos componentes, o tratamento de interrupções e o escalonamento de tarefas. Esta ferramenta possibilita que o projetista escreva uma rotina de verificação para um determinado componente. Quando a plataforma for executada, o TOS carrega a rotina escrita pelo projetista junto com as demais rotinas de verificação da plataforma.

Outra ferramenta disponível é o CoreConnect Test Generation (CTG). O CTG gera um conjunto de rotinas para verificar se um dado dispositivo é compatível com a arquitetura de barramento da plataforma. As rotinas automatizados são construídas a partir de algumas informações fornecidas pelo projetista, através de uma interface gráfica, favorecendo assim a sua usabilidade.

A plataforma ARM PrimeXsys [ALP 03] também possui um mecanismo semelhante para facilitar a verificação da interação de um componente com um barramento. Esta ferramenta denomina-se *AMBA Complience Test-bench* (ACT) [ALP 03]. O ACT gera um conjunto de rotinas para verificar a compatibilidade entre um determinado dispositivo e o barramento AMBA. Esta compatibilidade é garantida quando o dispositivo é verificado em uma lista de cenários-chave para o protocolo de comunicação.

A grande diferença entre CTG e ACT é que o primeiro opera num nível mais alto de abstração, enquanto o ACT é utilizando para o suporte à simulação RTL.

2.2 Outros mecanismos de suporte à verificação

Em [CAR 99] apresenta-se um método para construir sistemas funcionalmente insensíveis à latência de comunicação entre componentes pré-verificados. Para viabilizar a transmissão de dados com elevada latência entre módulos (por exemplo, acima de um ciclo de relógio), os autores propõem o uso de *relay stations*. Uma *relay station* é um *buffer* que atua como um *pipeline* em um determinado canal de comunicação, gerando sinais de *stall* (pausa) para paralisar os componentes do sistema enquanto realiza a transmissão de dados.

Existem algumas semelhanças entre a abordagem apresentada em [CAR 99] e a técnica proposta neste trabalho. Por exemplo, para analisar o fluxo de dados em um canal, os autores daquele trabalho desconsideram os sinais de *stall* gerados pelas *relay stations*. De forma similar, na abordagem proposta neste trabalho as entradas de *log* que correspondem a eventos gerados pelos artefatos de implementação são desconsideradas, pois estes eventos não exercem influência sobre o fluxo de dados no sistema.

No entanto, a técnica proposta neste trabalho (Capítulo 4) difere da abordagem proposta em [CAR 99] nos seguintes pontos:

- Os sinais de *stall* gerados pelas *relay stations* possuem uma marcação (*tag*) que os diferencia dos sinais válidos do sistema, enquanto as entradas de *log* produzidas pelos artefatos de implementação não possuem marcação alguma que as diferencie dos demais dados produzidos pelo sistema. Ou seja, a técnica aqui proposta não requer instrumentação para identificar artefatos de implementação, pois isto limitaria sua aplicabilidade em verificação.

- O método proposto em [CAR 99] supõe que os componentes do sistema sempre conservam a ordem dos dados, ao contrário da técnica proposta neste trabalho, que opera adequadamente mesmo quando o DUV não preserva a ordem dos comportamentos (uma das maiores dificuldades na verificação pós-particionamento segundo [GM 07])

- A observabilidade da técnica aqui proposta é maior do que a do método apresentado em [CAR 99], pois ela permite o acesso a elementos internos de um componente (*white-box*).

- A técnica proposta em [CAR 99] só pode ser aplicada em sistemas construídos com componentes que possuem a propriedade de serem paralisados através de um sinal de *stall*, uma limitação ao projeto da qual a técnica aqui proposta pode prescindir.

Em [ALB 07], a verificação *white-box* é viabilizada através do mecanismo de reflexão computacional. A contribuição-chave está no mecanismo de suporte minimamente intrusivo (o projetista não precisa fazer instrumentação adicional alguma no DUV, nem adicionar biblioteca alguma; os elementos internos do DUV podem ser inspecionados sem que seja necessário revelar o código-fonte). Embora essa técnica tenha sido cuidadosamente construída para permitir uma abordagem *white-box* minimamente intrusiva, seu potencial ainda não foi inteiramente explorado. Devido a estas vantagens, este trabalho utiliza o mecanismo proposto em [ALB 07] como infra-estrutura de implementação para inserção de monitores e para a captura de comportamentos, os quais serão automaticamente analisados pela técnica proposta nesta dissertação.

2.3 Abordagens para a não-preservação da ordem

Há pelo menos três abordagens convencionais para contornar o problema da não-preservação da ordem de comportamentos:

- **Uso de modelo de referência com precisão de ciclos** [GHE 05]: ao temporizar o modelo de referência em relação à implementação, a ordem seria preservada, mas estariam comprometidos o desempenho do modelo e a reusabilidade do *testbench*, conforme já discutido na Seção 1.6.

- **Ordenação baseada em antevisão do modelo RTL** [YIM 97]: toda vez que uma tomada de decisão se fizer necessária para o tratamento de eventos

concorrentes, uma amostra ("trace") da implementação RTL é sintetizada sob demanda para instrumentar a decisão de forma a escolher apropriadamente a ordem. Esta implementação pode, por exemplo, ser executada concorrentemente com o modelo funcional, recebendo os mesmos estímulos. A diferença entre os sinais de saída dos modelos funcional e RTL deve guiar os projetistas na ordenação de comportamentos. No entanto, ainda segundo [YIM 97], heurísticas são necessárias para diferenciar defeitos reais de artefatos de implementação, as quais podem limitar as garantias de verificação

- **Uso de scoreboard** [GM 07]: Um *scoreboard* é uma estrutura de dados que armazena resultados esperados (ou estímulos injetados) e que compara o resultado obtido no DUV com o resultado armazenado (ou transformado a partir dos estímulos armazenados). A Figura 2.1 ilustra a estrutura de um *scoreboard*.

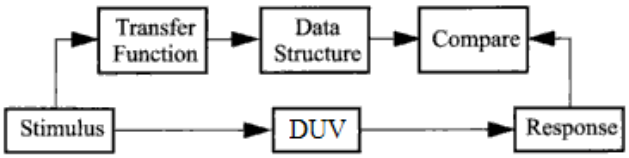

Figura 2.1: Estrutura de um *scoreboard* (extraído e adaptado de [BER 05])

Segundo [BER 05], um *scoreboard* pode tratar o problema da não-preservação de ordem através do módulo de comparação. Este módulo pode ser implementado de forma a lidar com discrepâncias temporais e de ordenamento entre os dados obtidos no DUV e aqueles armazenados na estrutura de dados. Por outro lado, um *scoreboard* não é capaz de verificar aspectos internos ao DUV [BER 05].

Além desta limitação espacial, o tratamento dos dados amostrados por um *scoreboard* possui uma limitação temporal. Ou seja, para cada dado amostrado no DUV o *scoreboard* deve efetuar uma decisão local para encontrar a entrada correspondente no modelo de referência. Esta decisão local está apoiada em

heurísticas que podem afetar a qualidade da verificação conforme será ilustrado a seguir.

Suponha uma variável, que ao ser implementada num modelo de mais baixo nível admita uma perda de precisão da ordem de 0.5. A Figura 2.2a ilustra uma possível execução deste modelo. Repare que, quando o valor 8 é amostrado à saída do DUV, o *scoreboard* precisa tomar uma decisão local para determinar qual das duas entradas amostradas no modelo de referência é a entrada correspondente ao valor 8 amostrado pelo DUV. Como o *scoreboard* não possui nenhuma informação a respeito de quais serão os próximos valores amostrados, a associação de entradas deve ser feita baseada numa heurística, conforme ilustrado na Figura 2.2b. No entanto, conforme apresentado na Figura 2.2c, a associação efetuada não está correta.

Em outras palavras, as limitações de visibilidade de um *scoreboard* sacrificam a qualidade da verificação em prol de uma maior eficiência computacional. O Capítulo 5 apresenta indícios dessa limitação no que diz respeito a qualidade da verificação.

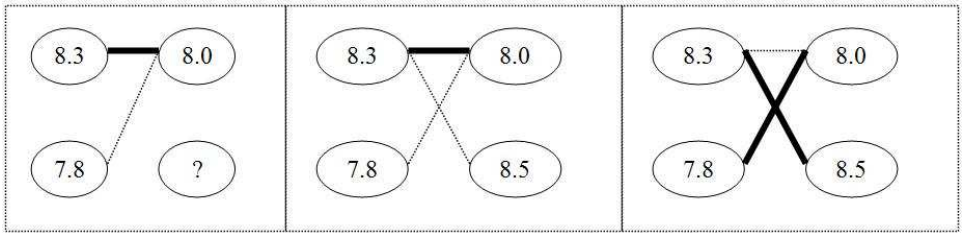

Figura 2.2: O impacto da limitação temporal na qualidade da verificação

2.4 A perspectiva de uma abordagem alternativa

Esta dissertação propõe uma abordagem alternativa para viabilizar a verificação de DUVs que possam não preservar a ordem dos comportamentos do modelo comportamental atemporal. A abordagem proposta não requer um modelo

temporizado, não requer a antevisão de um modelo RTL e provê o acesso aos elementos internos do DUV, o que amplia a observabilidade da verificação para além das possibilidades de um *scoreboard*.

O próximo capítulo descreve os fundamentos para a análise automática de *logs* (a ser proposta no Capítulo 4), formulando um problema de verificação funcional de um sistema como um problema de casamento em um grafo bipartido.

Entretanto, a construção de casamentos próprios deve levar em conta a compatibilidade de valores capturada no grafo e a causalidade de eventos à luz da relação de precedência de eventos extraída da especificação. Se não fosse especificada ordem alguma entre eventos, algoritmos clássicos poderiam ser reusados para encontrar casamentos. Entretanto, quando a ordem está especificada, casamentos não podem ser obtidos diretamente a partir de algoritmos clássicos, sob pena de ser violado o princípio da causalidade.

Capítulo 3

Modelagem do problema

3.1 Propriedades da relação entre RGM e DUV

Para estabelecer uma correlação funcional entre dois modelos, vamos utilizar três conceitos principais:

- Unicidade (para determinar que cada comportamento é preservado e exibido uma única vez),
- Compatibilidade (para determinar quais valores são aceitáveis),
- Causalidade (para garantir comportamento temporal adequado).

Antes de formular o problema-alvo é preciso formalizar algumas noções preliminares. Assuma que um engenheiro de verificação selecionou k pontos $m_1, m_2, ...m_k$ para monitorar em um módulo de uma plataforma. Vamos chamar estes pontos de *monitores*.

Toda vez que um monitor detecta uma alteração e um novo valor v, observado em um instante t_i, dizemos que ocorreu um *evento* v_i.

Assuma que todo valor observado num monitor é amostrado toda vez que é alterado, durante um intervalo de observação.

Definição 1: Log de um monitor. Vamos denotar como t_i (com $i \neq 0$) o tempo em que um elemento monitorado sofre a sua i-ésima transição e vamos cha-

mar de t_0 o momento em que o elemento recebe o seu valor inicial. O log de um monitor m_k é uma série de eventos caracterizada por uma seqüência de valores $\langle v_{k0}, v_{k1}, ..., v_{ki}...v_{kn}\rangle$ observados nos instantes $\langle t_0, t_1, ..., t_i, ...t_n\rangle$.

Quando os valores observados por um monitor m são atributos representados como números inteiros sinalizados ou não (por exemplo: `char`, `int`, `short`, `long`, `unsigned`, etc.) tanto no RGM como no DUV e desde que os valores monitorados por m' e m'' usem exatamente o mesmo número de bits em sua representação, uma relação de equivalência deve ser verificada para os valores monitorados por m, conforme formalizado abaixo.

Definição 2: Equivalência de valores. Dois valores u e v são equivalentes e denotados como $u \equiv v$, se e somente se $u = v$.

No caso de serem representados como números de diferentes tipos (por exemplo: `double` no RGM e `sc_fixed` no DUV) ou com diferente precisão (por exemplo: `sc_int<16>` no RGM e `sc_int<12>` no DUV), deve ser adotada uma relação de compatibilidade onde seja definida uma margem aceitável de ruído de quantização (por exemplo, se um atributo do tipo `double` é monitorado por m' no RGM e um atributo do tipo `sc_fixed<5,3>` é monitorado por m'' no DUV, então uma relação de compatibilidade deve adotar uma margem de quantização de 0,25, pois o atributo no DUV pode assumir qualquer valor x no intervalo -4,75 $\leq$ x $\leq$ 3,75 em incrementos de 0,25). Esta noção é formalizada a seguir.

Definição 3: Compatibilidade de valores. Dois valores u e v são compatíveis e denotados como $u \approx v$, se e somente se forem iguais ou se a sua diferença estiver dentro de uma margem pré-estabelecida.

Note que esta definição permite verificar, por exemplo, modelos abstratos contra modelos com precisão de *bits*.

Seja I^+ uma instância de um módulo em uma plataforma, usada como referência. Esta instância será chamada de *modelo de referência* (RGM). Seja I^- uma nova instância de um módulo, derivada de I^+ (através de um refinamento,

por exemplo). I^- será chamado de *dispositivo sob verificação* (DUV).

Vamos denotar por R a *relação de precedência entre eventos* ditada pela especificação, como formalizado a seguir:

Definição 4: Relação de precedência entre eventos no RGM. Vamos denotar por V^+ o conjunto de eventos monitorados no modelo de referência. A relação de precedência entre eventos é o conjunto $R = \{(v_i^+, v_j^+) \in V^+ \times V^+ : v_i^+ \text{ precede } v_j^+\}$.

A próxima definição utiliza as Definições 1, 3, 4 para estabelecer o conceito de mapeamento próprio. Um mapeamento próprio é utilizado para estabelecer a correlação entre dois modelos. Dois modelos serão considerados correlatos quando implementam todos os comportamentos especificados, guardando a compatibilidade de valores (a qual é capturada pela relação $\approx$) e respeitando a noção de causalidade (a qual é capturada pela relação R).

Definição 5: Mapeamento próprio. Dadas duas instâncias I^+ e I^- de um módulo, um monitor m e as seqüências de eventos $\langle v_0^+, v_1^+, ..., v_n^+ \rangle$ e $\langle v_0^-, v_1^-, ..., v_p^- \rangle$ observados para I^+ e I^-, respectivamente, diz-se que existe um *mapeamento próprio* de eventos, se e somente se existir um mapeamento μ: $V^+ \rightarrow V^-$ onde $V^+ = \{v_0^+, v_1^+, ..., v_n^+\}$ e $V^- = \{v_0^-, v_1^-, ..., v_p^-\}$ tal que todas as cláusulas abaixo seja satisfeitas:

- μ é uma injeção (completude e unicidade);
- $\mu(v_i^+) = v_j^- \Rightarrow v_i^+ \approx v_j^-$ (compatibilidade);
- $\forall v_i^+, v_x^+ : ((v_i^+, v_x^+) \in R) \wedge (\mu(v_i^+) = v_j^-) \wedge (\mu(v_x^+) = v_k^-) \Rightarrow j < k$ (causalidade)

Note que não é necessário que μ seja uma bijeção, já que as implementações em níveis mais baixos podem apresentar comportamentos não especificados (i.e. $|V^+| \leq |V^-|$). Note também que, se um mapeamento μ for encontrado, pode-se garantir que cada comportamento especificado no RGM está corretamente implementado no DUV, sob a perspectiva de um dado monitor.

Definição 6: Compatibilidade baseada em logs. O DUV I^- é funcionalmente compatível com o *golden model* I^+ no que diz respeito ao conjunto de monitores $\{m_k\}$, se e somente se existir um mapeamento próprio para cada monitor.

3.2 Formulação do problema-alvo

As definições apresentadas permitem formular, de forma sucinta, o problema-alvo:

Problema-Alvo: Dados dois modelos I^+ e I^-, verificar se eles são funcionalmente compatíveis no que diz respeito ao conjunto de monitores $\{m_k\}$.

Este problema-alvo pode ser reformulado como um problema de casamento em grafos bipartidos para cada monitor. Para isso, serão preliminarmente revisadas algumas noções clássicas, as quais serão personalizadas, em seguida, com informações específicas do problema-alvo.

Um grafo G(V, E) é dito *bipartido* se e somente se:

- $V = V^+ \cup V^-$ e $V^+ \cap V^- = \emptyset$;
- $\forall (u, v) \in E : (u \in V^+) \wedge (v \in V^-)$.

Um *casamento* (ou *matching*) de um grafo $G(V, E)$ é um subconjunto M de E tal que nenhum vértice é incidente a mais de uma aresta em M. Dizemos que um vértice u está *casado*, se existe uma aresta $(u, v) \in M$ ou *livre*, em caso contrário.

Um casamento com a cardinalidade máxima é denominado de *casamento máximo*. Se cada vértice de G incidir em uma aresta de M, então M será considerado um *casamento perfeito* (note que todo casamento perfeito é máximo, mas nem todo máximo é perfeito). Se cada vértice da partição V^+ incide em uma aresta de M, então M será denominado de *casamento completo*.

Um *componente conexo* $C_j(V_j, E_j)$ de um grafo bipartido G(V,E) é um subgrafo de G tal que, para todos $u, v \in V_j$, há um caminho entre u e v.

A noção de componente conexo corresponde à relação de equivalência "v é alcançável a partir de u". Como uma relação de equivalência é o mesmo que uma partição [COR 90], a decomposição de um grafo G(V,E) em termos de seus componentes conexos resulta em uma partição de G.

Por resultar em uma partição de G, quando um algoritmo de casamento garante um casamento completo de cada componente conexo, pode-se garantir o casamento completo do grafo G.

Definition 7: Grafo bipartido de verificação de um monitor. Sejam m^+ e m^- instâncias de um monitor m cujos logs são $\langle v_0^+, v_1^+, ..., v_n^+ \rangle$ e $\langle v_0^-, v_1^-, ..., v_p^- \rangle$. O *grafo bipartido de verificação*, denotado por BVG(m), é um grafo bipartido $G(V, E)$ onde:

- $V^+ = \{v_0^+, v_1^+, ..., v_n^+\}$, $V^- = \{v_0^-, v_1^-, ..., v_p^-\}$;
- Existe uma aresta $(v_i^+, v_j^-) \in E$, se e somente se $v_i^+ \approx v_j^-$.

Definição 8: Casamento próprio. Um casamento $\mathcal{M}$ de um BVG(m) será considerado *próprio*, se e somente se induz um mapeamento próprio μ.

Definição 9: Aresta imprópria. Uma aresta (v^+, v^-) é dita *imprópria* se e somente se $(v^+, v^-) \notin \mathcal{M}$.

Para abordar a não-preservação da ordem de comportamentos no DUV, a técnica aqui proposta baseia-se em uma noção fundamental, o assim-chamado cruzamento impróprio de arestas. Antes de formalizar esta importante noção, um exemplo ilustrativo é utilizado para introduzi-la.

As Figuras 3.1a 3.1b mostram a relação topológica entre duas arestas, sob uma mesma relação de precedência de eventos: o evento v_i^+ precede o evento v_x^+ no RGM. Assim, no cenário da Figura 3.1a, o cruzamento topológico entre as arestas (v_i^+, v_j^-) e (v_x^+, v_k^-) resultaria numa inversão de eventos no DUV que não obedeceria a precedência especificada. Portanto, nestas condições, tais arestas não

podem co-existir em um casamento próprio, pois representariam uma violação do princípio da causalidade. Por outro lado, a inexistência de cruzamento topológico entre as arestas, conforme ilustrado na Figura 3.1b, permite que ambas possam co-existir em um casamento próprio. Note que a mera existência de um cruzamento topológico não indica por si só a violação do princípio da causalidade, como mostra a Figura 3.1c: se a relação de precedência não especifica ordem alguma entre os eventos v_i^+ e v_x^+, ambas as arestas são elegíveis para um casamento próprio. Assim, nem todo cruzamento topológico resulta em cruzamento impróprio, conforme formalizado a seguir.

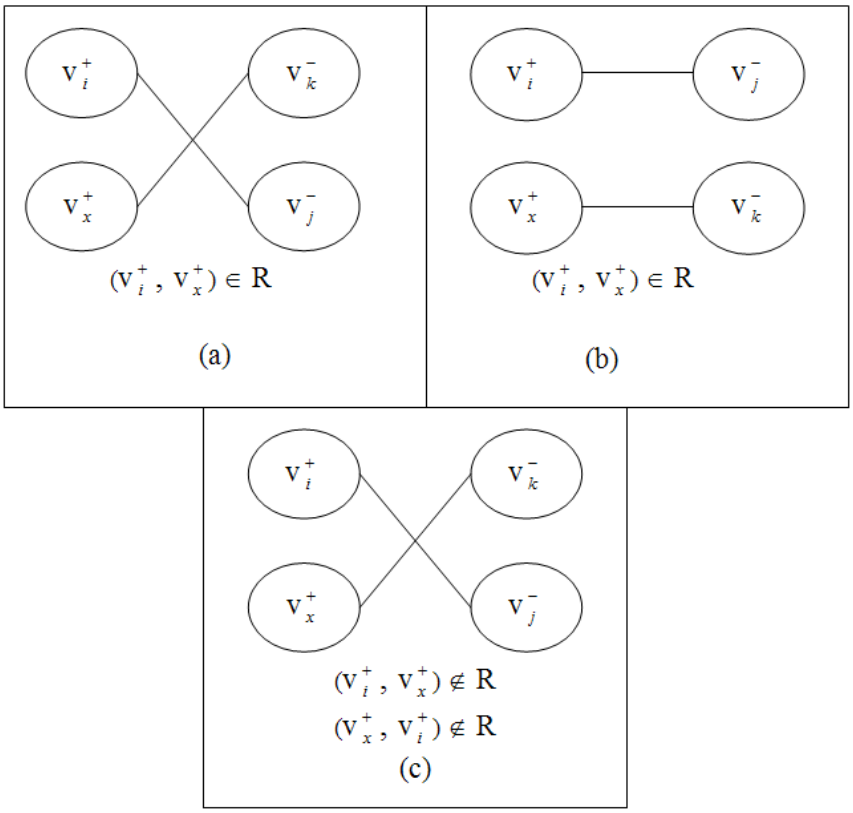

Figura 3.1: Correlação entre topologia e causalidade.

Definição 10: Função cruzamento. Dadas duas arestas (v_i^+, v_j^-) e (v_x^+, v_k^-), com $i \neq x$, sua *função cruzamento*, escrita $\chi((v_i^+, v_j^-), (v_x^+, v_k^-))$, retorna verdadeiro se e somente se: $(((v_i^+, v_x^+) \in R) \wedge (j > k) \vee ((v_x^+, v_i^+) \in R) \wedge (k > j)))$.

Definição 11: Cruzamento impróprio de arestas. Dizemos que existe um *cruzamento impróprio* entre duas arestas (v_i^+, v_j^-) e (v_x^+, v_k^-), com $i \neq x$, se e somente se $\chi((v_i^+, v_j^-), (v_x^+, v_k^-))$ retorna verdadeiro.

Note que, ao se detectar um cruzamento impróprio, não se pode decidir qual das arestas envolvidas é imprópria. Somente quando uma delas está garantidamente no casamento $\mathcal{M}$, a outra torna-se imprópria. Assim, a noção de cruzamento impróprio permitirá identificar e eliminar arestas não elegíveis para um casamento próprio, assim que for detectada que uma delas é própria.

3.3 Garantias teóricas decorrentes da formulação

As noções apresentadas nas seções anteriores viabilizam a prova de algumas propriedades que serão utilizadas durante a construção de um casamento próprio.

Teorema 1: Identificação de aresta imprópria através de cruzamento impróprio. Sejam um casamento próprio $\mathcal{M}$ e uma aresta $(v_i^+, v_j^-) \in \mathcal{M}$. Se existe um vértice v_x^+ tal que $(v_i^+, v_x^+) \in R$ e há um cruzamento impróprio entre as arestas (v_x^+, v_k^-) e (v_i^+, v_j^-), então a aresta (v_x^+, v_k^-) é imprópria.

Prova: Se há cruzamento impróprio e o evento v_i^+ precede o evento v_x^+, concluímos da Definição 10 que j > k, o que contraria a definição de mapeamento próprio μ (terceira condição da Definição 5). Logo, pela Definição 8, conclui-se que $(v_x^+, v_k^-) \notin \mathcal{M}$ e esta aresta é, portanto, imprópria. □

Note que o Teorema 1 assume que $\mathcal{M}$ existe e que $(v_i^+, v_j^-) \in \mathcal{M}$. Assim, a identificação de arestas impróprias só pode ser realizada a partir de uma aresta já casada (e ainda assim sob a hipótese de que esta aresta pertence a $\mathcal{M}$). Ou seja, o Teorema 1 deve ser aplicado incrementalmente à medida que os vértices são visitados (*pruning* dinâmico), mas não poderia ser aplicado a priori (*pruning* estático).

Teorema 2: Identificação de aresta compulsória. Se existe um casamento $\mathcal{M}$

e há uma única aresta (v^+, v^-) incidente em v^+, então $(v^+, v^-) \in \mathcal{M}$ e esta aresta é dita compulsória.

Prova: Como (v^+, v^-) é única, ela deve pertencer a um casamento completo (se existir um), o que é condição necessária para a existência de um casamento $\mathcal{M}$ (do contrário μ não seria uma função). □

Cabe ressaltar que o Teorema 2 é um corolário do assim-chamado "Teorema da Redução de Grau Um" [KAR 81], quando aplicado ao problema do casamento completo.

Note ainda que, ao contrário do Teorema 1, o Teorema 2 pode ser aplicado a priori, ou seja, antes de se iniciar quaisquer visitas a vértices para promover seu casamento (*pruning* estático).

Teorema 3: Identificação de aresta imprópria induzida por aresta compulsória. Se (v^+, v^-) é uma aresta compulsória, então todas as demais arestas incidentes em v^- são impróprias.

Prova: Como $(v^+, v^-) \in \mathcal{M}$, pela Definição 8, temos $\mu(v^+) = v^-$. Ora, para que uma aresta arbitrária (u^+, v^-) com $u^+ \neq v^+$ fosse própria, deveríamos ter $\mu(u^+) = v^-$, o que contrariaria a primeira condição da Definição 5 (não-injeção). Logo, toda aresta (u^+, v^-) é imprópria. □

Note que, o Teorema 3 não pode ser aplicado isoladamente, mas pode ser invocado imediatamente após a aplicação do Teorema 2 (que representa portanto uma pré-condição).

Vamos enunciar uma importante propriedade de grafos bipartidos, que será invocada no algoritmo a ser proposto, para avaliar a existência de um casamento.

Teorema 4: Teorema de Hall: Seja um grafo bipartido G(V,E) cujas partições são V^+ e V^-. Sejam S $\subseteq V^+$ e Adj(S) = $\{v^- \in V^- \mid ((v^+, v^-) \in E) \wedge (v^+ \in S)\}$. Existe um casamento completo M, se e somente se $|S| \leq |Adj(S)|$ para todo S $\subseteq V^+$.

Prova: Disponível em [HAL 56] □.

Por resultar em uma partição do BVG, a noção de componente conexo é central para o algoritmo a ser proposto. Uma primeira utilidade desta noção é formalizada a seguir.

Definição 11: Componente conexo impróprio. Um componente conexo $C_j(V_j, E_j)$ de um grafo bipartido G(V,E) é dito *impróprio*, se e somente se o grafo $C_j(V_j, E_j)$ não satisfaz o Teorema de Hall.

Teorema 5: Identificação de componente conexo impróprio. Dado um componente conexo $C_j(V_j, E_j)$ de um grafo bipartido G(V,E), se $|V_j^+| > |V_j^-|$ então não existe um casamento próprio para G.

Prova: Para S = V_j^+, pela definição de componente conexo, tem-se Adj(S) = V_j^-. Como por hipótese C_j é impróprio, tem-se |S| > |Adj(S)|, o que contraria a condição necessária para a existência de um casamento completo dos vértices da partição V^+, segundo o Teorema de Hall. Ora, isso viola a primeira cláusula da Definição 5 e, pela Definição 8, não há casamento próprio em G. □

Note que o Teorema 5 estabelece uma condição suficiente para identificar um componente conexo impróprio. Em outras palavras, podem existir componentes conexos impróprios não identificáveis através deste teorema.

Assim, para a observabilidade de um dado conjunto de monitores, a aplicação do Teorema 5 garante que nenhum defeito existente no DUV é negligenciado (ausência de falso negativo). Por outro lado, o Teorema 5 por si só não garante que um casamento completo é encontrado sempre que existir um (potencial de falso positivo). Conclui-se assim que esse teorema tem a virtude de restringir o espaço de pesquisa, mas como sua capacidade de diferenciar artefatos de implementação de defeitos é limitada, não pode ser usado como única garantia para a verificação de um sistema. Uma discussão detalhada do impacto do Teorema 5 sobre a qualidade da verificação é apresentada na Seção 4.2.

Lembre, que pelos Teoremas 1 e 3, quando uma aresta (v^+, v^-) é

adicionada a um casamento, outras arestas podem ser removidas por serem consideradas impróprias. Se esta remoção de arestas tornar impróprio um componente conexo originalmente próprio, então (v^+, v^-) não pode pertencer a um casamento próprio. A Figura 3.2 ilustra esta situação. Para o grafo da Figura 3.2a, suponha que, $(v_i^+, v_j^+) \in R$ e $(v_i^+, v_k^+) \in R$. Repare que nele não há arestas compulsórias e, aparentemente, todas as arestas parecem igualmente elegíveis para um casamento. Suponha que a aresta (v_i^+, v_n^-) seja selecionada para o casamento. Ora, neste caso as arestas (v_i^+, v_l^-), (v_j^+, v_m^-) e (v_k^+, v_m^-) tornam-se impróprias (a primeira torna-se imprópria pela primeira cláusula de Definição 5, e as duas últimas pelo Teorema 1). A Figura 3.2b ilustra o cenário resultante de se ter selecionado (v_i^+, v_n^-) para $\mathcal{M}$ (assumindo-a como própria), onde a aresta selecionada é destacada e as arestas removidas estão pontilhadas. A remoção das arestas pontilhadas induziria o componente conexo mostrado na Figura 3.2c, que é impróprio. Conclui-se portanto que (v_i^+, v_n^-) é imprópria e não deve ser incluída em $\mathcal{M}$.

Os próximos dois teoremas formalizam as noções ilustradas.

Teorema 6: Identificação de arestas impróprias através de componentes conexos impróprios. Sejam um grafo bipartido G(V,E) e uma aresta $(v^+, v^-) \in E$. Seja E' o conjunto de arestas que resultam em cruzamento impróprio com (v^+, v^-). Se o grafo reduzido $G'(V, E - E')$ possui um componente conexo $C_j(V_j, E_j)$ tal que $|V_j^+| > |V_j^-|$, então (v^+, v^-) é imprópria.

Prova: Suponha, por absurdo, que (v^+, v^-) fosse própria; isto é, $(v^+, v^-) \in \mathcal{M}$. Pelo Teorema 1, toda aresta $(v_x^+, v_k^-) \in E'$ seria imprópria, ou seja, $E' \cap \mathcal{M} = \emptyset$. Logo, nessa condição teríamos $\mathcal{M} \subseteq E - E'$, ou seja, ao supor (v^+, v^-) própria, haveria garantia de existência de um casamento $\mathcal{M}$ em G'. Entretanto, como por hipótese G' possui um componente $C_j(V_j, E_j)$ tal que $|V_j^+| > |V_j^-|$, o Teorema 5 garante (ao contrário) a inexistência de um casamento $\mathcal{M}$ em G'. Desta contradição conclui-se que (v^+, v^-) é, na verdade, imprópria. □

Teorema 7: Identificação de arestas impróprias pelo princípio da unicidade de comportamentos. Sejam um grafo bipartido G(V,E) e uma aresta

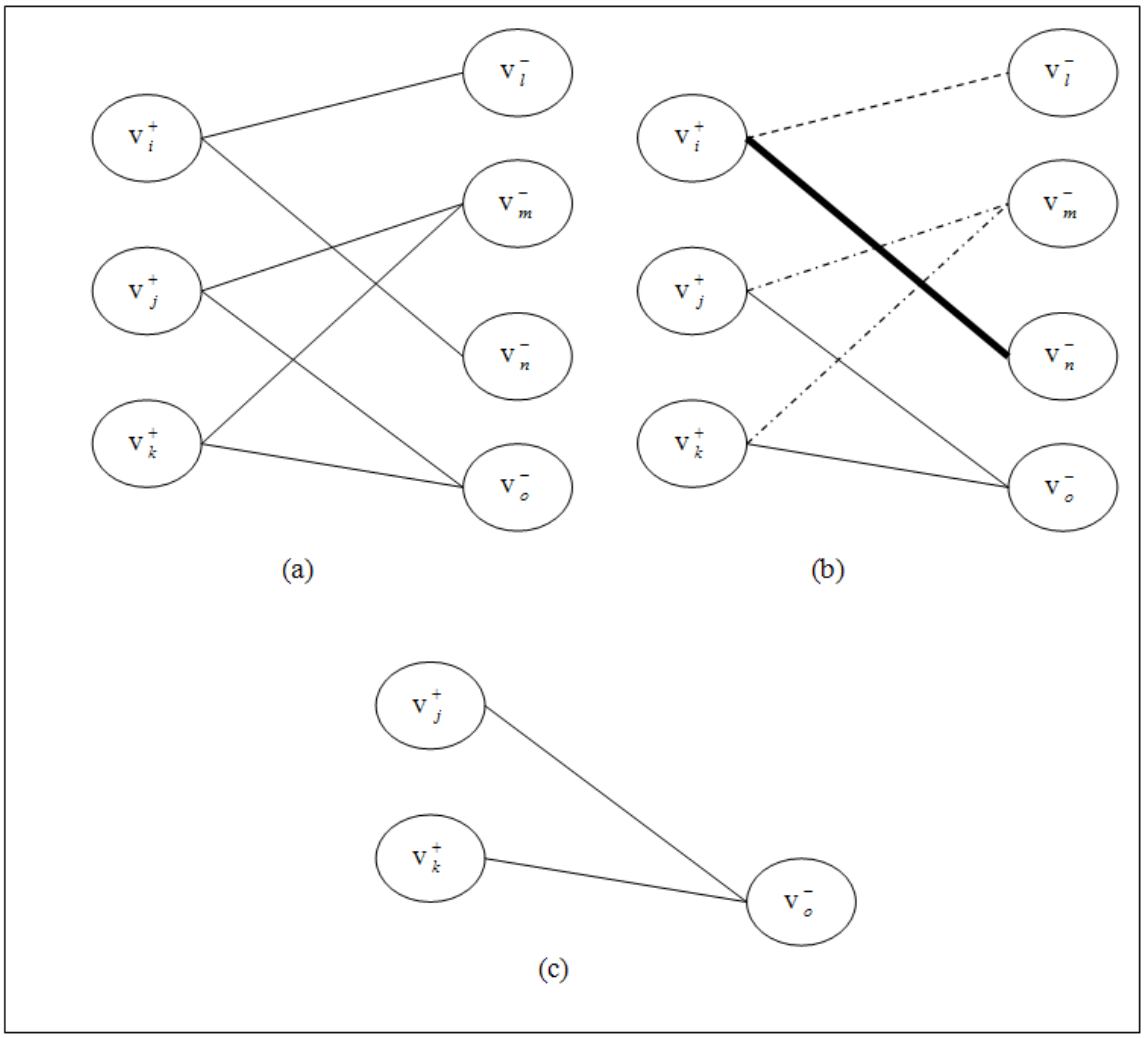

Figura 3.2: Componente conexo impróprio induzido por aresta imprópria

$(v_i^+, v_k^-) \in E$. Seja E' o conjunto de arestas incidentes a v_k^- e diferentes de (v_i^+, v_k^-). Se o grafo reduzido $G'(V, E - E')$ possui um componente conexo $C_j(V_j, E_j)$ tal que $|V_j^+| > |V_j^-|$, então (v_i^+, v_k^-) é imprópria.

Prova: Suponha, por absurdo, que (v_i^+, v_k^-) fosse própria; isto é, $(v_i^+, v_k^-) \in \mathcal{M}$. Pela primeira cláusula da Definição 5, cada aresta $(v_x^+, v_k^-) \in E$ seria imprópria, ou seja, $E' \cap \mathcal{M} = \emptyset$. O restante da prova é exatamente idêntico ao da prova do Teorema 6. □

Um importante corolário do Teorema de Hall, formalizado a seguir,

será utilizado (no próximo capítulo) para discutir as garantias de verificação da técnica proposta nesta dissertação.

Teorema 8: Corolário do Teorema de Hall para relações de equivalência: Seja G(V,E) um grafo cujas partições são V^+ e V^- e onde E é induzido por uma relação de equivalência $\equiv$. Existe um casamento completo M para G se e somente se $|V_j^+| \leq |V_j^-|$ para cada componente $C_j(V_j, E_j)$ de G.

Prova: Suponha um componente conexo $C_j(V_j, E_j)$. Por definição, para cada par de vértices $v_k^+ \in V_j^+$ e $v_z^- \in V_j^-$ existe um caminho $v_k^+, v_l^-, v_m^+, v_n^-, v_o^+, v_p^- ..., v_z^-$. Como as arestas refletem uma relação transitiva $\equiv$, então podemos afirmar que existem arestas (v_k^+,v_n^-), (v_k^+,v_p^-), (v_k^+,v_z^-). Em outras palavras, Adj(v_k^+) = V_j^-. Logo, se $|V_j^+| \leq |V_j^-|$), então para qualquer conjunto S, tal que $S \subseteq V_j^+$, temos que $|S| \leq |Adj(S)|$, o que satisfaz o Teorema de Hall. □

No próximo capítulo, as noções aqui formalizadas são utilizadas para descrever os algoritmos propostos e para estabelecer suas garantias de verificação.

Capítulo 4

A técnica de verificação proposta

Algoritmos eficientes para encontrar casamentos em grafos bipartidos estão disponíveis na literatura [HOP 73] [ALT 91]. No entanto, estes algoritmos não são adequados para resolver diretamente o problema-alvo, porque não capturam a noção de causalidade entre eventos. Assim, embora os algoritmos clássicos pudessem encontrar um casamento máximo para um dado grafo, as arestas escolhidas para um casamento poderiam violar a ordem de eventos ditada pela especificação.

4.1 O algoritmo proposto

Por simplicidade, todos os algoritmos apresentados nesta seção assumem que V, E, R e M são conjuntos com escopo global.

Antes de uma aresta (v_i^+, v_j^-) ser incluída em um casamento $\mathcal{M}$, todos os vértices v_x^+ tais que $(v_i^+, v_x^+) \in R$ devem ser previamente examinados para verificar a existência ou não de cruzamento impróprio com arestas já casadas, conforme mostrado no Algoritmo 1, o qual aplica a Definição 10 iterativamente.

O Algoritmo 2 captura a mera identificação de arestas compulsórias, como definido no Teorema 2.

Algorithm 1 `causal(`(v_i^+, v_j^-)`)`

1: **for all** v_x^+ **such that** $(v_i^+, v_x^+) \in R$ **do**
2: **for all** $(v_x^+, v_k^-) \in$ M **do**
3: **if** $\chi((v_i^+, v_j^-)\ ,\ (v_x^+, v_k^-))$ **then**
4: return false
5: **end if**
6: **end for**
7: **end for**
8: return true

Algorithm 2 `compulsory-edges()`

1: COMP $\leftarrow \emptyset$
2: **for all** $v^+ \in V^+$ **do**
3: **if** $|\text{Adj}(v^+)| = 1$ **then**
4: COMP $\leftarrow$ COMP $\cup \{(v^+, v^-)\}$
5: **end if**
6: **end for**
7: return COMP

Dada uma aresta identificada como compulsória, o Algoritmo 3 aplica o Teorema 2 e a inclui no casamento (linha 2), caso essa aresta seja causal (linha 1). Assim, se uma aresta compulsória não puder ser casada, o casamento não será completo, e portanto não será próprio; neste caso, o algoritmo retorna um predicado falso. Ao serem casadas, arestas compulsórias induzem arestas impróprias, conforme os Teoremas 1 e 3. Portanto, quando uma aresta compulsória é incluída no casamento, removem-se as arestas que se tornaram impróprias após a inclusão de (v^+, v^-) em M (linhas 3 e 4) (estas arestas removidas podem, inclusive, ser arestas compulsórias, induzindo um casamento incompleto, o que seria um indicador de não compatibilidade funcional).

A simplicidade dos Algoritmos 4 e 5 dispensa explicações, exceto pelo fato de que a função `remove-improper-edge(`v^+, v^-`)` não se limita a remover

Algorithm 3 `match-compulsory`$((v^+, v^-))$

1: **if** causal$((v^+, v^-))$ **then**
2: M $\leftarrow$ M $\cup$ $\{(v^+, v^-)\}$
// Theorem 2
3: remove-improper-edges-crossing$((v^+, v^-))$
// Theorem 1
4: remove-improper-edges-incident-to(v^-)
// Theorem 3
5: P $\leftarrow$ true
6: **else**
7: P $\leftarrow$ false
8: **end if**
9: return P

Algorithm 4 `remove-improper-edges-incident-to`(v_j^-)

1: **for all** $v^+ \in V^+$ such that $(v^+, v^-) \in E$ **do**
2: `remove-improper-edge`$((v^+, v_j^-))$
3: **end for**

Algorithm 5 `remove-improper-edges-crossing`$((v_i^+, v_j^-))$

1: **for all** $(v_x^+, v_k^-) \in E$ such that improper-crossing$((v_i^+, v_j^-), (v_x^+, v_k^-))$ **do**
2: `remove-improper-edge`$((v_x^+, v_k^-))$
3: **end for**

Algorithm 6 `remove-improper-edge`$((v^+, v^-))$

1: E $\leftarrow$ E - $\{(v^+, v^-)\}$
2: **if** $|(\text{Adj}(v^+)| = 1)$ **then**
3: **let** $\{u^-\} = \text{Adj}(v^+)$
4: `match-compulsory`$((v^+, u^-))$
5: **end if**

a aresta (v^+, v^-), conforme mostra o Algoritmo 6: quando uma aresta imprópria é removida (linha 1), ela pode tornar compulsória uma outra aresta (linha 2). Por

isso, neste caso, o Algoritmo 6 invoca (à linha 4) o Algoritmo 3 (já descrito), que se torna recursivo.

Dado um conjunto de arestas compulsórias, o Algoritmo 7 as inclui no casamento (se forem causais) e retorna o predicado adequado.

Algorithm 7 `match-compulsory(COMP)`

```
1: for all (v+, v−) ∈ COMP do
2:    P ← match-compulsory((v+, v−))  // Theorem 2
3:    if P = false then
4:       return false
5:    end if
6: end for
7: return true
```

Algorithm 8 `prune-improper-edges()`

```
1: COMP ← compulsory-edges()
   // Theorems 1 and 3
2: P ← match-compulsory(COMP)
3: if P then
4:    for all (v+, v−) ∈ E do
5:       if implies-improper-edge((v+, v−)) then
6:          // Theorems 6 and 7
            E ← E - {(v+, v−)}
7:          P ← prune-improper-edges()
8:       end if
9:    end for
10: end if
11: return P
```

O Algoritmo 8 remove do conjunto E todas as arestas consideradas impróprias. Esta identificação está baseada nos Teoremas 1, 3, 6 e 7. Nas linhas 1 e 2, ele determina as arestas compulsórias causais e as inclui no casamento (lembre que,

ao fazê-lo, as arestas consideradas impróprias pelos Teoremas 1 e 3 são removidas conforme o Algoritmo 3 já descrito). Repare que, quando se detecta que uma aresta é imprópria pelo critério do Teorema 6 (linha 5) a função `prune-improper-edges` é recursivamente invocada (linha 7). Isto é necessário porque a remoção de uma aresta (v^+, v^-) pode tornar outras arestas compulsórias ou impróprias. Ao final deste processo recursivo de remoção, todas as arestas remanescentes em E estão casadas ou são elegíveis para o casamento M.

A função `implies-improper-edge` é descrita pelo Algoritmo 9. Nas linhas 1 e 2 o Algoritmo 9 obtém todas as arestas que não podem ser incluídas em um casamento M se a aresta (v_i^+, v_j^-) estiver em M, pelos critérios dos Teoremas 6 e 7. Para isto (na linha 1) obtém-se o conjunto E' contendo todas as arestas que resultam em cruzamento impróprio com (v_i^+, v_j^-), ou seja, arestas não-causais. Em seguida (na linha 2) obtém-se o conjunto E'' de arestas incidentes a v_j^+ que não satisfazem a primeira cláusula da Definição 5 ou seja, que resultam em não-unicidade de comportamentos.

Algorithm 9 `implies-improper-edge`$((v_i^+, v_j^-))$

1: $E' \leftarrow \{(v_x^+, v_k^-) \in E | \chi((v_i^+, v_j^-), (v_x^+, v_k^-))\}$
2: $E'' \leftarrow \{(v_y^+, v_j^-) \in E | y \neq i\}$
3: $E^* \leftarrow$ E - $(E' \cup E'')$
4: **for all** $(v_x^+, v_k^-) \in (E' \cup E'')$ **do**
5: **if** implies-improper-component(v_x^+, E^*) **then**
6: // Theorems 6 and 7
return true
7: **end if**
8: **end for**
9: return false

Na linha 3, o Algoritmo 9 obtém o conjunto reduzido de arestas E*, dele excluindo as arestas que violariam o princípio da causalidade (E') e as arestas que resultariam em não-injeção (E'').

Repare que, na linha 4, o Algoritmo 9 visita somente as arestas (v^+, v^-) afetadas pela potencial inclusão de (v_i^+, v_j^-) no casamento $(E' \cup E'')$ e não sobre todas as arestas em E. Isto faz com que apenas os componentes conexos efetivamente afetados sejam verificados (linha 5).

Tudo se passa como se a aresta (v_i^+, v_j^-) fosse especulativamente incluída no casamento e todas as arestas impróprias fossem especulativamente removidas. Sob esta perspectiva o algoritmo verifica se algum dos componentes conexos afetados pela inclusão especulativa de (v_i^+, v_j^-) tornou-se impróprio (linha 5). Em caso positivo, o algoritmo encerra a busca, retornando um predicado verdadeiro (ou seja, a aresta (v_i^+, v_j^-) é imprópria), caso contrário, ele retorna falso. Note que a verificação de um dado componente conexo (linha 5) é feita utilizando o conjunto reduzido de arestas E*, que seria resultante desta especulação.

O Algoritmo 10 verifica se a redução no conjunto de arestas tornaria impróprio o componente conexo que inclui um vértice v^+. O algoritmo supõe a existência de uma função `implies-improper(`C_j`),` que verifica se um dado componente conexo é impróprio pelos critérios dos Teoremas 6 e 7.

Algorithm 10 `implies-improper-component(`v^+, E^*`)`

1: let $G' = (V, E^*)$
2: let C_j be the connected component of G' including v^+
3: return `implies-improper(`C_j`)` // Theorems 6 and 7

Finalmente, o Algoritmo 11 invoca os algoritmos anteriormente descritos para obter um casamento próprio M para o BVG, se existir um (M = $\mathcal{M}$), ou retorna um predicado falso, em caso contrário. Após eliminar (à linha 2) as arestas impróprias detectadas a priori (relembre que esta ação pode resultar na inclusão de arestas compulsórias no casamento M), o algoritmo verifica se a existência de um casamento próprio foi detectada (P = true) ou não (P = false). No segundo caso, o algoritmo retorna um predicado falso e o conjunto de arestas incluídas no casamento incompleto (que podem ser usadas para fins de depuração). No primeiro caso, o algoritmo visita todos os vértices ainda não casados da partição V^+, a fim

de selecionar arestas neles incidentes para o casamento M (invocando à linha 5, o Algoritmo 12).

Algorithm 11 `proper-matching()`

```
1: M ← ∅
2: P ← prune-improper-edges()
3: if P then
4:    for all v_i^+ ∈ V^+ such that v_i^+ is unmatched do
5:       P ← match(v_i^+)
6:       if P = false then
7:          return (P, M)
8:       end if
9:    end for
10: end if
11: return (P, M)
```

Algorithm 12 `match(`v_i^+`)`

```
1: P ← false
2: if ∃(v_i^+, v_j^-) ∈ E then
3:    M ← M ∪ {(v_i^+, v_j^-)}
4:    P ← true
5:    E' ← {(v_x^+, v_k^-) ∈ E | χ((v_i^+, v_j^-), (v_x^+, v_k^-))}
6:    E'' ← {(v_y^+, v_j^-) ∈ E | y ≠ i}
7:    if (E' ≠ ∅) ∨ (E'' ≠ ∅) then
8:       E ← E - (E' ∪ E'')
9:       P ← prune-improper-edges()
10:   end if
11: end if
12: return P
```

O Algoritmo 12 inclui um vértice v^+ no casamento. Relembre que todas as arestas que se pôde detectar a priori como impróprias já foram removidas

(na linha 2 do Algoritmo 11), logo, todas as arestas presentes em E estão em M, ou são igualmente elegíveis para o casamento M (ou seja, nenhuma das arestas restantes em E leva a uma violação de causalidade com as arestas já presentes em M). A linha 2 verifica se o conjunto E ainda possui alguma aresta incidente ao vértice v_i^+, incluindo-a (se existir) no casamento M (relembre que todas as arestas incidentes a v_i^+ são igualmente elegíveis). Uma vez incluída uma aresta (v_i^+, v_j^-) no casamento, todas as arestas que causem violações de causalidade (linha 5) com (v_i^+, v_j^-) ou que resultem em não-unicidade de comportamentos (linha 6) são removidas do grafo. Relembre que a aresta (v_i^+, v_j^-) já foi submetida ao Algoritmo 9 e, portanto, já se verificou que a remoção das arestas em $E' \cup E''$ não inviabiliza o casamento de componente conexo algum. Repare ainda que se não existir uma aresta incidente a $v_i^+ \in E$ então o algoritmo retornará um predicado falso e, consequentemente, o Algoritmo 11 (na linha 7) retornará um predicado falso e um casamento incompleto.

Finalmente, repare que, quando uma aresta (v_i^+, v_j^-), é adicionada ao casamento, todas as demais arestas incidentes a v_j^- são removidas do grafo (linha 6). Isto garante, que numa futura execução do Algoritmo 12 para um dado vértice v_k^+, qualquer aresta incidente a v_k^+, selecionada na linha 2, incide sobre um vértice v_l^- não casado (afinal, se v_l^+ tivesse, por hipótese, sido casado com um vértice v_x^+ então a aresta (v_k^+, v_l^-) teria sido removida, durante a execução do Algoritmo 12 para o vértice v_x^+). Assim, obrigatoriamente, para toda aresta $(v_i^+, v_j^-) \in G$, se v_i^+ não está casado, então v_j^- também não está.

4.2 Garantias de verificação da técnica proposta

4.2.1 Análise de falso negativo

Vamos agora analisar a possibilidade de o Algoritmo 11 não detectar um erro no DUV, ou seja de induzir um falso negativo.

Isto corresponderia ao cenário onde o Algoritmo 11 retorna P = true, mas o casamento M encontrado (aparentemente próprio) viola a Definição 5

(M $\neq \mathcal{M}$). O Algoritmo 11 retorna P = true somente se um casamento completo é encontrado (Algoritmo 11, linha 5). Como M é completo e as arestas do BVG capturam por construção a relação de compatibilidade de valores, o algoritmo retornaria um M impróprio associado a um predicado P = true somente se a terceira cláusula da Definição 5 (causalidade) fosse violada. Isto aconteceria se uma aresta imprópria, digamos (v_i^+, v_j^-), fosse incluída em M embora causasse um cruzamento impróprio com alguma outra aresta anteriormente incluída em M, digamos (v_x^+, v_k^-). Assim, vamos analisar a possibilidade de inclusão de arestas impróprias, a qual poderia ser realizada apenas por duas funções invocadas pelo Algoritmo 11.

A função `match-compulsory` inclui uma aresta em M (Algoritmo 3, linha 2) sob a pré-condição `causal` = true (linha 1), que só ocorre quando $\forall (v_i^+, v_x^+) \in R, (v_x^+, v_k^-) \in M : \chi((v_i^+, v_j^-), (v_x^+, v_k^-)) =$ false (Algoritmo 1), ou seja, esta função nunca inclui arestas que violem a terceira cláusula da Definição 5.

A função `match` inclui uma aresta em M (Algoritmo 12, linha 3) sob a pré-condição `prune-improper-edges` = true (Algoritmo 11, linha 5), que só ocorre quando $\forall (v^+, v^-) \in M$: `match-compulsory` (v^+, v^-) = true (Algoritmo 7, linha 2), que por sua vez se mantém quando $\forall (v^+, v^-) \in M$: `causal` (v^+, v^-) = true (Algoritmo 3, linha 1), uma condição que garante a não-violação da Definição 5, conforme provado anteriormente.

Isto demonstra que nenhuma aresta imprópria é incluída em M sob o predicado P = true e, desta forma, o Algoritmo 11 nunca resulta num falso negativo. Ou seja, quando um casamento M associado a um predicado P = true é obtido, então M = $\mathcal{M}$ e a técnica proposta garante que não existe erro algum no DUV no que diz respeito ao comportamento capturado por um dado monitor.

4.2.2 Análise de falso positivo

Vamos agora analisar o possibilidade de o Algoritmo 11 induzir falsos positivos, ou seja, detectar um erro aparente no DUV que na verdade não existe. Isto corresponderia ao caso onde o Algoritmo 11 retorne um casamento

incompleto M sob o predicado P = false, embora exista um casamento causal $\mathcal{M}$ que não foi encontrado.

Neste cenário, existe uma aresta, digamos (v_i^+, v_j^-) não incluída em M, mas cuja inclusão teria preservado todas as cláusulas da Definição 5 (uma aresta própria). Isto poderia acontecer se uma aresta imprópria, digamos (v_x^+, v_k^-) fosse erroneamente incluída em M e, devido a um cruzamento impróprio com a aresta (v_i^+, v_j^-) esta foi removida de E e, desta forma, de M.

Assim, vamos analisar a possibilidade de uma aresta própria ser excluída. O Algoritmo 11 invoca apenas duas funções que excluem arestas: `match-compulsory` (Algoritmo 3, linhas 3 e 4) e `prune-improper-edges` (Algoritmo 8, linha 6).

Primeiramente vamos analisar a função `match-compulsory`. Note que o Algoritmo 3 é invocado (no Algoritmo 6 à linha 4 e no Algoritmo 8 à linha 2) sempre sob a pré-condição de que (v^+, v^-) foi identificada como uma aresta compulsória pelo Teorema 2. Como o Teorema 2 estabelece uma condição necessária e suficiente para uma dada aresta ser compulsória (ou seja, não existe a chance de uma aresta ser erroneamente considerada compulsória), o Algorimo 3 apenas remove arestas que não poderiam co-existir com as arestas compulsórias num casamento próprio. Assim, a função `match-compulsory` nunca exclui arestas próprias.

Vamos agora analisar a função `prune-improper-edges`. O Algoritmo 8 exclui arestas não apenas à linha 2 através da função `match-compulsory` (já analisada), mas também à linha 6, onde arestas são removidas pelos critérios dos Teoremas 6 e 7, que estão encapsulados na função `implies-improper-edge` (Algoritmo 9). Já que estes teoremas estabelecem uma condição suficiente (mas não necessária) para detectar uma aresta como própria, arestas impróprias podem ser erroneamente tomadas por próprias (Algoritmo 9, linha 5) e permanecer em E, tornando-se assim candidatas para M. Se alguma destas arestas for incluída em M, ela induzirá a remoção de arestas próprias de E.

Para evitar manter arestas impróprias em E, o Algoritmo 8 deveria usar (na linha 5) a condição necessária e suficiente estabelecida pelo Teorema de

Hall (Teorema 4), que é computacionalmente ineficiente porque requer a análise de todos os subconjuntos de vértices da partição V^+.

Entretanto, é importante notar que, embora os Teoremas 6 e 7 estabeleçam condições suficientes (mas não necessárias) para a detecção de arestas impróprias quando operam sob uma relação de compatibilidade $\approx$, estas condições tornam-se *necessárias* e suficientes, de acordo com o Teorema 8, quando operam sob uma relação de equivalência $\equiv$ entre RGM e DUV (devido à transitividade).

4.2.3 Discussão

O algoritmo proposto reduz o esforço necessário para um engenheiro de verificação depurar um sistema. Isto acontece porque, para um dado conjunto de monitores, o engenheiro pode confiar que todos os casamentos causais encontrados estão corretos e não refletem um falso negativo (conforme foi discutido na Seção 4.2.1). No entanto, o engenheiro de verificação precisará verificar todos os monitores que não encontraram casamentos causais, pois alguns deles podem resultar num falso positivo conforme discutido em 4.2.2.

Assim, embora um falso positivo não possa ser completamente evitado para uma relação arbitrária de compatibilidade (a menos que o custo da análise de todos os subconjuntos seja aceitável), o Teorema 8 garante que nenhuma aresta própria é excluída quando o BVG representa uma relação de equivalência. Portanto, para uma relação de equivalência, o Algoritmo 11 nunca resulta em falso positivo. Como pode-se esperar que uma relação de equivalência representa um caso prático freqüente (por exemplo quando um atributo do tipo inteiro é monitorado), conclui-se que o *overhead* de se avaliar todos os componentes conexos através do Teorema de Hall deva ser evitado. Por outro lado, embora possível, a ocorrência de falso positivos não é necessariamente freqüente e pode provavelmente ser reduzida através da escolha judiciosa de heurísticas para seleção de arestas, mas sem limitar o espaço de pesquisa.

De qualquer maneira, ao reduzir o espaço de pesquisa necessário

para o engenheiro de verificação, o algoritmo proposto reduz o esforço necessário para diferenciar artefatos de implementação de reais defeitos do sistema. Esta diferenciação, segundo [YIM 97] não é uma tarefa trivial, e normalmente requer o uso de heurísticas para sua realização.

Note que, se existir mais do que um casamento próprio, o Algoritmo 11 encontra apenas um deles. Entretanto, como se provou que o algoritmo não resulta em falso negativo, o fato de existir mais do que um mapeamento próprio não desqualifica a verificação. Ou seja, embora o algoritmo não distingua qual dos comportamentos implementados efetivamente corresponde ao especificado, esta distinção só faria sentido se o algoritmo fosse usado para depuração. Todavia, não faria sentido fazer-se a depuração de um projeto cuja verificação não detectou erro algum.

É fácil verificar que a maioria dos algoritmos propostos limita-se a visitar elementos do conjunto V^+ uma única vez. Assim, tais algoritmos realizariam, no pior caso, $O(|V^+|)$ operações. Entretanto, dois algoritmos em particular exigem uma inspeção mais detalhada para avaliação de complexidade.

O Algoritmo 1 efetua uma varredura para detectar se a inclusão de uma aresta (v_i^+, v_j^-) induz alguma violação de causalidade no casamento obtido. Para isso, no laço iniciado na linha 1, ele obtém todos os vértices que possuem uma relação de precedência com v_i^+ e verifica se alguma aresta nele incidente já foi incluída no casamento M (linha 2). Repare que, como cada vértice possui uma ou nenhuma aresta em M, o laço que se inicia na linha 2 vai iterar no máximo uma vez. Desta forma, no pior caso, o Algoritmo 1 realizaria $O(|V^+|)$ operações.

O Algoritmo 8 é o gargalo da técnica proposta, visto que se utiliza de uma função recursiva (`prune-improper-edges`). Seja N_i o número de arestas remanescentes no grafo quando essa função é invocada pela i-ésima vez. A i-ésima invocação requer a visita a cada uma das N_i arestas para se efetuar a verificação à linha 5. A execução da linha 5 do Algoritmo 8 demanda um máximo de N_i operações a cada aresta visitada. Assim, no pior caso, cada invocação requer $N_i \times N_i$ operações. Ora, na primeira invocação da função recursiva existem $N_1 = |E|$ arestas, o que

requer $|E| \times |E|$ ou $|E|^2$ operações. No entanto, como uma aresta é eliminada (à linha 6) a cada nova invocação, a função recursiva será invocada no máximo $|E|$ vezes e, em sua última invocação, haverá uma única aresta a ser visitada ($N_{|E|} = 1$), ou seja, uma única operação a ser realizada. Em outras palavras, o número total de operações é a soma de uma progressão aritmética cujo primeiro elemento é $|E|^2$, cujo último elemento é 1 e cujo número de elementos é $|E|$. Portanto, no pior caso, o número de operações é $|E| \times (|E|^2 + 1)/2 = (|E|^3 + |E|)/2$. Ou seja, a complexidade de pior caso do Algoritmo 8 é $O((|E|^3 + |E|)/2) \leq O(|E|^3)$.

O próximo capítulo discute aspectos da implementação e apresenta resultados experimentais como evidência de validação da técnica proposta.

Capítulo 5

Implementação e resultados experimentais

5.1 Implementação

Por não impactar o desempenho da técnica proposta, o mecanismo de captura de entradas de *log* foi implementado em Java. Por outro lado, o mecanismo de análise de *logs*, que corresponde aos Algoritmos de 1 a 12, foi implementado em C++. Para a implementação dos grafos, utilizou-se uma codificação eficiente para as listas de adjacência, conforme sugerido em [WEI 99].

Os algoritmos apresentados na Seção 4.1 foram descritos visando esclarecer sua funcionalidade e explicitar a relação de suas ações com a base teórica descrita no Capítulo 3 para fins de estabelecer garantias formais. Ao implementá-los algumas otimizações foram feitas para fins de desempenho.

Uma importante otimização foi feita no Algoritmo 9. Repare que o laço que inicia na linha 4 é aplicado para cada vértice v_x^+ impactado pela adição especulativa de (v_i^+, v_j^-). Desta forma, para cada vértice v_x^+ impactado, o algoritmo descrito constrói e avalia o componente conexo correspondente. Ora, se dois vértices impactados, digamos v_x^+ e v_z^+ pertencem a um mesmo componente conexo $C_j(V_j, E_j)$, então se o componente conexo de v_x^+ já foi analisado, não há razão para repetir a

análise para v_z^+. Assim, ao determinar que um componente conexo é próprio, o algoritmo implementado armazena os vértices em V_j^+ numa cache local, para evitar pesquisas desnecessárias. Como resultado desta otimização, o Algoritmo 9 visita cada vértice no máximo uma vez, o que resulta numa complexidade $O(|V|)$ no pior caso.

Uma outra importante otimização foi utilizada para melhorar o desempenho do Algoritmo 8. Note que aquele algoritmo invoca iterativamente o Algoritmo 9 à linha 5, através da rotina `implies-improper-edge`, o que resulta em uma complexidade $O(|V|^2)$ no pior caso para o Algoritmo 8. Entretanto, como o Algoritmo 9 foi construído de forma a não alterar nenhum dos conjuntos globais (V, E e M), cada invocação da rotina `implies-improper-edge` é totalmente independente de outras invocações. Assim, cada uma dessas invocações pode ser associada a uma *thread* de execução distinta, potencializando assim sua execução em paralelo e reduzindo o impacto da complexidade algorítmica sobre o tempo de execução.

Uma segunda otimização foi introduzida no Algoritmo 8. Para amplificar o impacto da filtragem de arestas impróprias, foi introduzida uma heurística que potencializa a eliminação de um maior número de arestas a cada iteração, sem limitar o espaço de pesquisa, conforme descrito a seguir.

Note que, no Algoritmo 3 (que é invocado à linha 2 do Algoritmo 8), quando uma aresta compulsória é encontrada tem-se o potencial de eliminar um grande número de arestas (linhas 3 e 4). Assim, ao conduzir a verificação de forma a encontrar novas arestas compulsórias, o processo de filtragem pode ser amplificado levando a um melhor desempenho. Como a eliminação de uma aresta (v_i^+, v_j^-) pode tornar compulsória outra aresta (v_i^+, v_k^-), após eliminar uma aresta incidente em um vértice v_i^+, é conveniente analisar outras arestas incidentes no mesmo vértice. Assim, no laço delimitado pelas linhas 4 e 9 do Algoritmo 8, induziu-se uma ordem de verificação de arestas tal que todas as arestas incidentes a um mesmo vértice v_i^+ são analisadas antes de se passar às arestas incidentes em outros vértices. Note que, como o Algoritmo 8 assume uma ordem arbitrária de pesquisa de arestas, o ordenamento realizado pela heurística adotada não limita o espaço de pesquisa,

salvaguardando as garantias de verificação da Seção 4.2

Todos os experimentos de validação reportados nesta dissertação foram executados em um processador Intel Xeon com 4 núcleos, operando à freqûência de 2,66 GHz, com 4GB de memória principal (SDRAM, DDR-2, 667MHz). Nos experimentos, todos os núcleos de processamento disponíveis foram utilizados para permitir a verificação de arestas em paralelo.

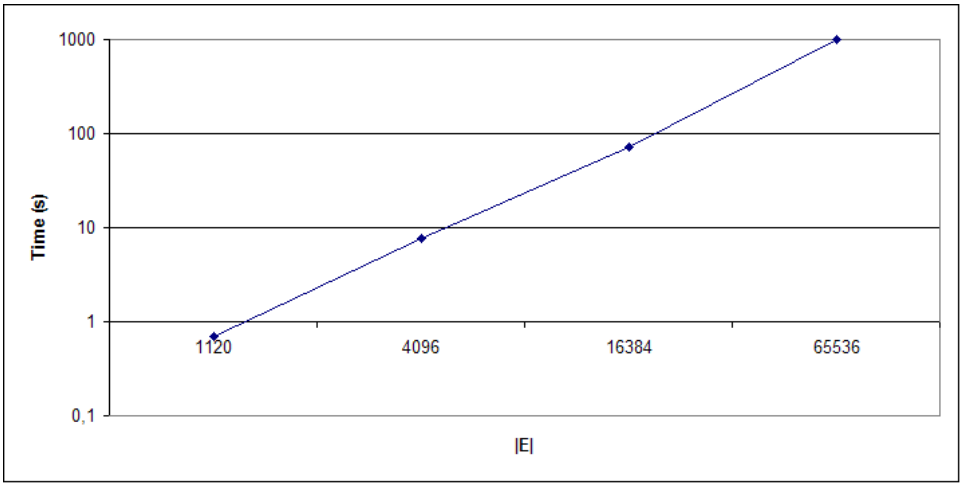

Figura 5.1: Análise do tempo de execução.

Como evidência preliminar de que a complexidade do caso médio do algoritmo é bastante inferior à de pior caso, foi efetuado o seguinte experimento: foram construídos quatro grafos bipartidos completos, com número de vértices |V|, 2|V|, 4|V| e 8|V| (onde V = $V^+ \cup V^-$) (ao dobrar o número de vértices, o número de arestas |E| é quadruplicado). Cada um dos grafos foi submetido à técnica proposta sob a condição de que todos os vértices da partição V^+ estavam ordenados pela relação de precedência R.

A Figura 5.1 mostra o tempo de execução como função de $|E|$. Observe que a razão entre o tempo de execução e o número de arestas é aproximadamente constante (o tempo é multiplicado por 10, conforme o número de arestas é multiplicado por 4). Esse crescimento linear do tempo de execução com o número de arestas indica uma baixa complexidade média dos algoritmos propostos.

Na próxima seção será reportada a validação experimental da funcionalidade dos algoritmos descritos no Capítulo 4.

5.2 Resultados experimentais

Além das garantias teóricas apresentadas no Capítulo 4, foram realizados experimentos para fins de validação experimental, em três cenários distintos.

O primeiro cenário procura exercitar a técnica proposta em uma situação um tanto improvável na prática, na qual o RGM não especifica ordenação alguma entre eventos. Esta situação de mínima restrição temporal corresponde ao único cenário em que algoritmos clássicos de casamento poderiam ser diretamente reusados para resolver o problema (que, neste caso, corresponde a se encontrar um casamento completo que respeite a compatibilidade de valores).

No segundo cenário, o objetivo é o de validar a técnica proposta na situação extrema em que o RGM estabelece ordenação total de eventos. Embora este cenário seja também improvável na prática, ele tem a virtude de exercitar a técnica aqui proposta no extremo oposto do cenário anterior: a situação de máxima restrição temporal (neste caso, a ordem em que os vértices devem ser casados é única). Neste caso, embora algoritmos clássicos de casamento não possam ser diretamente aplicados, a ordenação total de eventos determina a ordem em que os vértices são casados, o que poderia ser feito através de um algoritmo bastante simples.

Finalmente, o terceiro cenário exercita a técnica proposta na situação prática mais freqüente em que o RGM estabelece uma ordenação parcial de eventos. Neste cenário, algoritmos clássicos de casamento não poderiam ser aplicados, pois resolveriam apenas o problema da compatibilidade de valores, sem respeitar a ordenação parcial. É justamente este o cenário que justifica a elaboração de um algoritmo que resolva simultaneamente a compatibilidade de valores e a precedência temporal, tal como o proposto no Capítulo 4.

Como os dois primeiros cenários induzem um grande número de arestas compulsórias durante a filtragem de arestas impróprias, a execução da técnica

de verificação proposta resulta em tempos de execução desprezíveis (da ordem de décimos de segundo). Assim, serão reportados tempos de execução apenas para o terceiro cenário, cuja complexidade resulta em maior esforço computacional.

5.2.1 Infra-estrutura e configuração experimental

Todos os IPs utilizados nos experimentos foram descritos como módulos SystemC e desenvolvidos para a plataforma de referência ArchC (ARP: *ArchC Reference Platform*)[ARC 06].

A infra-estrutura de reflexão computacional utilizada para amparar a técnica aqui descrita foi desenvolvida em um trabalho correlato [ALB 07].

Nos Cenários 1 e 2 serão verificados IPs inseridos em uma representação executável de uma plataforma contendo um processador PowerPC, uma memória e um IP. Cada instância de plataforma executa distintas aplicações-alvo, escolhidas a partir do benchmark Mibench [GUT 01]. O particionamento hardware-software que resultou na definição de cada IP levou em conta três critérios principais:

- Tempo de execução: Rotinas com um maior tempo de execução foram preferencialmente escolhidas para implementação em hardware.

- Posição da rotina na árvore de chamadas: Rotinas-folha, ou seja, que não invocam outras rotinas, foram preferencialmente escolhidas por sua tendência em minimizar o tráfego no barramento.

- Número de vezes que uma rotina é invocada: Rotinas com um menor número de chamadas tendem a reduzir o tráfego no barramento.

Uma vez escolhida a rotina a ser implementada em hardware, o restante da aplicação-alvo foi implementada em software a ser executado no processador *PowerPC*.

Duas representações distintas foram utilizadas para cada IP. Para os DUVs escolheram-se IPs representados com precisão de ciclos. Para os modelos de

referência escolheram-se descrições puramente funcionais, sem nenhuma informação de temporização.

O primeiro IP submetido a verificação implementa a função F do cifrador Blowfish [SCH 94]. Esta função implementa a operação de substituição presente em todos os cifradores que implementam o modelo proposto por Horst Feistel [FEI 73].

O segundo IP implementa uma operação no algoritmo SHA-0 que implementa a função Hash [MER 90]. A família dos algoritmos de Hash é utilizada para compor um resumo (*digest*) de 160 bits de uma mensagem de entrada.

O terceiro IP implementa a operação de DCT (transformada discreta do cosseno) usada no algoritmo JPEG, que é um método comum para compressão de imagens.

O quarto IP implementa a transformada inversa do cosseno utilizada no algoritmo de MP3, que é um método para compressão de áudio.

5.2.2 Cenário 1: eventos não ordenados

A Tabela 5.1 apresenta os resultados do experimento. Na primeira coluna tem-se o nome do estudo de caso, na segunda o número de arestas do BVG, nas terceira e quarta colunas, respectivamente, a cardinalidade das partições V^+ e V^-, na quinta coluna o número de arestas removidas pelo *pruning* (Teoremas 1, 3, 6 e 7), na sexta a cardinalidade do casamento encontrado e, na última coluna, o predicado retornado pelo algoritmo.

Tabela 5.1: Cenário 1 - Casamento encontrado

estudo de caso	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$\|Pruned\|$	|M|	P
Blowfish	8398	8367	25104	0	8367	true
SHA-0	4873	4873	4873	0	4873	true
JPEG encoding	685	37	37	0	37	true
MP3	2666	2632	2632	0	2632	true

Observe que o algoritmo retornou um predicado verdadeiro ($P = true$) para a verificação de todos os IPs, o que significa que todos os comportamentos especificados foram reproduzidos pelo DUV ($|M| = |V^+|$). Para a primeira aplicação, note que o número de comportamentos não especificados ($|V^-| - |V^+|$) realizados pelo DUV é maior que o número de comportamentos especificados no modelo de referência ($|V^+|$). Isto indica que alguns artefatos de implementação foram introduzidos como resultado de refinamentos de projeto. Para as outras aplicações nenhum comportamento não especificado foi introduzido durante refinamento dos modelos no que diz respeito aos monitores observados.

Repare também que, em nenhum dos casos, o algoritmo conseguiu eliminar arestas. Isto acontece porque, como nenhuma ordem foi especificada para os comportamentos, nenhuma aresta pôde induzir o algoritmo a encontrar um casamento que violasse a causalidade.

Para verificar se a técnica proposta é efetivamente capaz de encontrar defeitos, algumas falhas foram intencionalmente inseridas nos DUVs. Na primeira plataforma (que implementa o cifrador Blowfish), um defeito foi inserido nas assim-chamadas caixas-S (S-boxes). As caixas-S são estruturas de dados consultadas pela função F em cada uma das suas invocações. Na segunda plataforma (que implementa a função de Hash e contem o IP SHA-0), um defeito foi inserido na função que determina quantas vezes o IP sha-transform deve ser acionado. O defeito foi construído para que o IP fosse acionado no DUV uma vez a menos do que no modelo de referência. No terceiro modelo, a incialização da variável monitorada foi alterada para receber o dobro do valor que deveria receber. E finalmente, no quarto modelo 3 diferentes defeitos foram inseridos; no primeiro defeito o multiplicador-acumulador foi alterado para apenas multiplicar, no segundo defeito o multiplexador de saída está invertendo os *frames* das camadas 2 e 3 e finalmente o terceiro defeito inseriu uma falha de sincronização que faz com que os *frames* seja amostrados um ciclo após do que deveriam.

A Tabela 5.2 mostra os resultados obtidos quando os defeitos foram inseridos nos DUVs.

Tabela 5.2: Cenário 1 - Casamento completo não encontrado

estudo de caso	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$\|Pruned\|$	$\|M\|$	P
Blowfish	80	8367	25104	0	80	false
SHA-0	4872	4873	4872	0	4872	false
JPEG encoding	361	37	37	0	19	false
MP3 (defeito 1)	52	2632	1813	0	4	false
MP3 (defeito 2)	308	2632	1819	0	18	false
MP3 (defeito 3)	2869	2632	2628	0	2559	false

Note que, como conseqüência dos defeitos introduzidos, nenhum dos DUVs conseguiu reproduzir todos os comportamentos especificados ($|M| < |V^+|$); desta forma, nenhum passou na verificação ($P =$ false). No primeiro IP, o Algoritmo proposto encontrou um casamento de tamanho 80. A análise do casamento obtido revelou que apenas as 80 primeiras entradas do conjunto $|V^+|$ possuem entradas correspondentes no conjunto $|V^-|$; ora, isto indica que um erro ocorreu durante a 81º invocação do IP. Como cada iteração do IP recebe como entrada o resultado da iteração anterior e consulta um valor na caixa-S, temos que o defeito deve estar inserido na caixa-S, já que o valor da 80º invocação do DUV é o mesmo valor obtido na 80º invocação do modelo de referência (de fato, lembre que o defeito foi inserido justamente na caixa-S). Repare que como a função F é não-linear, a partir do momento em que o IP retorna um valor errado no 81º acionamento, os valores observados no DUV sempre divergem dos valores observados no modelo de referência.

No segundo IP, o casamento resultante indica que uma única entrada do conjunto $|V^+|$ não encontrou uma entrada compatível no conjunto $|V^-|$. A análise do casamento resultante revela que esta entrada é exatamente a última. Isto deixa duas possibilidades principais de defeito: ou o IP foi acionado uma vez a menos do que deveria ou o último acionamento do IP produziu um valor não compatível com a última entrada de *log* obtida no modelo de referência. Ora, repare que o DUV possui uma entrada a menos de *log* do que o modelo de referência ($|V^+| - |V^-| = 1$),

o que indicaria ao engenheiro de verificação que o defeito existente provavelmente está localizado no controlador do IP.

No terceiro IP, o grafo construído mostra que vários comportamentos especificados foram observados (E= 361) mesmo após a inserção da falta já descrita. Isto acontece porque em várias iterações do IP a variável monitorada é inicializada com 0, e neste caso a falta inserida ao altera o valor da inicialização. Para as inicializações da variável com valor diferente de 0, não foi possível identificar um comportamento equivalente no DUV.

Finalmente, no quarto IP, repare que a severidade da falta afeta a topologia do grafo obtido. Enquanto no defeito 1 o grafo resultante possui apenas 52 arestas e o casamento obtido tem cardinalidade 4, o defeito 3 (que representa uma falha muito mais sutil) apresenta um grafo resultante com 2869 arestas e um casamento de cardinalidade 2559 (relembre que, para este caso, um casamento completo deveria possuir cardinalidade 2632).

Ou seja, em todos os casos os casamentos obtidos podem servir como ferramenta para auxiliar o engenheiro de verificação a encontrar uma falha num modelo.

5.2.3 Cenário 2: eventos totalmente ordenados

A Tabela 5.3 apresenta, para cada caso de estudo, a comparação entre a representação funcional e a representação com precisão de ciclos dos IPs escolhidos.

Tabela 5.3: Cenário 2 - Casamento próprio encontrado

estudo de caso	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$\|Pruned\|$	|M|	P
Blowfish	8398	8367	25104	31	8367	true
SHA-0	4873	4873	4873	0	4873	true
JPEG encoding	685	37	37	648	37	true
MP3	2666	2632	2632	34	2632	true

Repare que os resultados obtidos neste experimento são semelhantes aos resultados apresentados na Tabela 5.1, exceto pelos valores apresentados na quinta coluna. Para o segundo IP, o algoritmo não conseguiu remover aresta alguma. Isto se deve ao fato de que o casamento encontrado tem cardinalidade igual ao número de arestas presentes no grafo ($|E| = |M|$), o que significa que todas as arestas eram compulsórias e nenhuma aresta pôde ser removida. Já nos experimentos com o primeiro, terceiro e quarto IPs as arestas removidas indicam que o BVG construído possui arestas que induziam violações de causalidade no casamento completo obtido. A remoção destas arestas garantiu que o algoritmo convergisse para uma solução que respeitasse a causalidade (casamento próprio).

Repare ainda que $|Pruned| = |E| - |M|$, pois só existe um casamento próprio M para os monitores apresentados (ao contrário dos experimentos realizados no Cenário 1, onde qualquer casamento completo estava de acordo com a especificação). Logo, como M é único, toda aresta não presente em M é imprópria e o fato de o algoritmo ter localizado todas elas é mais uma evidência de sua correção.

Para verificar se a técnica proposta é capaz de encontrar defeitos, os IPs foram submetidos aos mesmos defeitos induzidos no Cenário 1.

A Tabela 5.4 mostra os resultados obtidos quando os defeitos foram inseridas nos DUVs.

Tabela 5.4: Cenário 2 - Casamento próprio não encontrado.

estudo de caso	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$\|Pruned\|$	|M|	P
Blowfish	80	8367	25104	0	80	false
SHA-0	4872	4873	4872	0	4872	false
JPEG encoding	361	37	37	342	19	false
MP3 (defeito 1)	52	2632	1813	48	4	false
MP3 (defeito 2)	308	2632	1819	301	7	false
MP3 (defeito 3)	2869	2632	2628	310	2559	false

Note que os resultados são semelhantes aos resultados exibidos na

Tabela 5.2. Isto acontece porque, como não foi possível encontrar um casamento completo no Cenário 1, fatalmente não seria possível encontrar um casamento próprio no Cenário 2.

Embora a técnica proposta seja certamente mais útil para encontrar defeitos que não podem ser facilmente identificados logo após as primeiras entradas de *logs*, os experimentos mostram que ela obviamente também captura comportamentos induzidos por faltas severas. Em todos os casos, os casamentos resultantes podem fornecer informações para ajudar o projetista com o diagnóstico de defeitos.

5.2.4 Cenário 3: eventos parcialmente ordenados

Neste cenário, para modelo de referência, escolheu-se uma descrição atemporal e puramente funcional de uma seqüência de instruções aritméticas a ser executada em um *core* hipotético com emissão estática de uma instrução por ciclo (*static single issue*). Esse core hipotético implementa quatro tipos de instruções aritméticas (adição, subtração, multiplicação e divisão) codificadas segundo um formato de 3 operandos similar ao das instruções do processador MIPS, adaptadas para um banco de 16 registradores de uso geral.

Para fazer o papel de DUV, escolheu-se uma representação para o *core* que utiliza uma versão simplificada do Algoritmo de Tomasulo [PAT 07] (*dynamic single issue*), de forma a induzir execução fora de ordem. Essa representação com precisão de ciclos, assume que adição e subtração tenham latências de 1 ciclo e que as instruções de multiplicação e divisão tenham latência de 4 ciclos. Desta forma, devido às diferentes latências, a ordem de execução das instruções será alterada no DUV.

O código utilizado para estimular o RGM e o DUV consiste de um laço contendo 22 instruções, o qual realiza 20 iterações, resultando em 440 instruções executadas. Note que, neste caso, a relação de precedência entre eventos é a relação de dependência de dados e de controle associada ao código.

A Tabela 5.5 mostra os resultados obtidos para a verificação entre

os eventos monitorados no RGM e no DUV. A primeira linha mostra os resultados obtidos quando monitores espelhados são inseridos à porta de escrita do banco de registradores. A segunda linha reporta resultados quando um monitor é inserido à saída da unidade de emissão (*Dispatch Unit*) no RGM e no DUV.

Tabela 5.5: Cenário 3 - Casamento próprio encontrado

Monitor	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$\|Pruned\|$	|M|	P	Tempo (s)
Register Write Port	4136	440	440	3696	440	true	8,5
Dispatch Unit	15200	440	440	14760	440	true	29

Note que a técnica proposta encontrou um casamento próprio para cada par de monitores, ou seja, conseguiu detectar que todas as inversões de ordem de eventos efetuadas pelo DUV não caracterizam violações de causalidade.

Repare que, o número de eventos é exatamente o mesmo (440) no RGM e no DUV, pois independentemente dos refinamentos de projeto introduzidos, cada instrução é executada uma única vez e dispara um único evento de escrita no banco de registradores, embora a ordem em que os eventos são disparados possa variar.

Note também que, novamente o número de arestas removidas pelo *pruning* é igual ao número total de arestas subtraído do número de arestas presentes no casamento encontrado ($|Pruned| = |E| - |M|$). Neste caso, assim como nos resultados exibidos na Tabela 5.3, esta igualdade é uma conseqüência do fato de que, embora seja possível encontrar múltiplos casamentos completos para o grafo em questão, somente um destes casamentos é próprio. Este valor contrasta com aqueles exibidos na Tabela 5.1 onde todo casamento completo é uma solução válida.

Para verificar se o método proposto é capaz de encontrar incompatibilidades entre o modelo de referência e o DUV alguns defeitos foram intencionalmente inseridos no DUV. O primeiro defeito foi inserido na rotina que verifica se uma instrução está apta para execução. Relembre que, no Algoritmo de Tomasulo, a execução de uma instrução só é disparada quando todas as suas dependências já fo-

ram resolvidas. O defeito inserido tornou todas as instruções aptas para a execução, ou seja, nenhuma verificação de precedência de instruções foi realizada.

A Tabela 5.6 ilustra a comparação entre os *logs* do modelo de referência e do DUV obtidos após a inserção do defeito no DUV.

Tabela 5.6: Cenário 3 - Casamento próprio não encontrado (caso 1)

Monitor	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$Pruned$	|M|	P	Tempo (s)
Register Write	3550	440	440	3483	67	false	8,3
Dispatch Unit	15200	440	440	14824	383	false	30

Repare que, em ambos os casos, não foi possível encontrar um casamento completo ($|M| < |V^+|$). É interessante observar que, embora todas as instruções tenham sido disparadas ($|V^-| = |V^+|$), a ordem de execução de algumas instruções contraria a ordem ditada pela especificação. Além disso, a alteração imprópria na ordem de execução resultou em hazards RAW e WAR, o que alterou substancialmente os valores armazenados no banco de registradores, resultando em um casamento de cardinalidade bastante baixa ao se monitorar a porta de escrita do banco de registradores ($|\mathrm{M}| = 67$).

Quanto ao número de arestas removidos pelo *pruning*, repare novamente que $|Pruning| = |E| - |M|$, mesmo quando um casamento próprio não é encontrado. Ou seja, o algoritmo encerrou a sua busca quando todas as arestas restantes no grafo eram inelegíveis.

Repare ainda que o defeito introduzido fez com que um maior número de arestas fossem consideradas impróprias pela terceira cláusula da Definição 5 (causalidade) que se tornaram impróprias pelo Teorema 1. Este fato eliminou a possibilidade de um casamento próprio ser encontrado. Este fato explica o aumento no número total de arestas removidas no experiento relacionado com o monitor Dispatch Unit em relação ao experimento exibido na Tabela 5.5. O número de arestas removidas no monitor no banco de registradores foi menor em relação ao experimento da Tabela 5.5 porque o número total de arestas também é menor, em relação

ao mesmo experimento. Isto acontece porque a inversão na ordem de execução de algumas instruções fez com que valores diferentes fossem escritos no banco de registradores, o que reduziu o número de vértices compatíveis, e consequentemente o número de arestas.

Para ilustrar como a utilização de múltiplos monitores poderia reduzir o esforço de verificação, outro defeito foi inserido no DUV: o operador de multiplicação foi alterado para somar, ao invés de multiplicar os operandos.

A Tabela 5.7 ilustra os resultados obtidos

Tabela 5.7: Cenário 3 - Casamento próprio não encontrado (caso 2)

Monitor	$\|E\|$	$\|V^+\|$	$\|V^-\|$	$Pruned$	|M|	P	Tempo(s)
Register Write	9	440	440	2	7	false	0.1
Dispatch Unit	15200	440	440	14760	440	true	29

Repare que o predicado retornado é verdadeiro para a monitoração de instruções, enquanto o predicado retornado é falso para a monitoração da porta de escrita do banco de registradores. Ou seja, os casamentos obtidos reduzem o esforço de verificação porque indicam que todas as instruções foram corretamente disparadas, mas que os valores escritos no banco de registradores não foram os esperados. Ora, isso indica que o defeito reside entre o momento em que uma instrução é enviada para execução (o que resulta num evento capturado pelo monitor à saída da *Dispatch Unit*) e o momento em que a instrução finaliza a execução (o que dispara uma escrita no monitor na porta de escrita no banco de registradores). De fato, o defeito foi inserido dentro da unidade de execução da operação de multiplicação.

Relembre que a técnica proposta neste trabalho pode induzir falsos positivos (ou seja, encontrar aparentes anomalias em sistemas funcionalmente equivalentes), conforme discutido na Seção 4.2.2. No entanto, para comprovar que a técnica proposta não retorna falso negativo (ou seja, não encontrar uma anomalia entre modelos funcionalmente não-equivalentes), uma verificação extra foi realizada após cada experimento: compararam-se os conteúdos do banco de registradores ob-

tidos no DUV e no modelo de referência após a execução do código. No experimento referente à Tabela 5.5, os conteúdos dos bancos de registradores resultaram idênticos. Nos experimentos relativos às Tabelas 5.6 e 5.7 os conteúdos dos bancos de registradores não resultaram idênticos, o que foi corretamente diagnosticado pelos casamentos incompletos resultantes.

5.3 Discussão

Como a técnica de *scoreboard* é talvez a abordagem convencional para tratar a não-preservação de ordem, realizou-se um experimento para apontar suas deficiências em relação à técnica aqui proposta. O experimento realizado consiste em dois somadores, que realizam operações em paralelo e escrevem o resultado num *buffer* conforme a Figura 5.2.

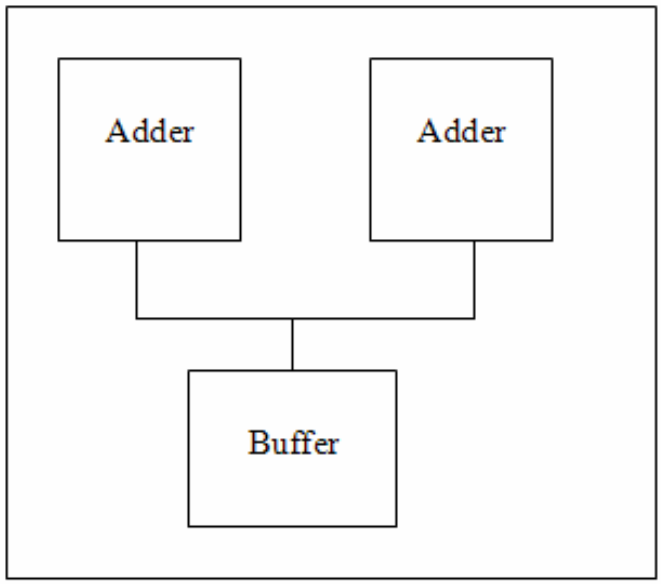

Figura 5.2: Estrutura do experimento.

Suponha que, numa implementação de baixo nível do modelo, as latências dos somadores foram alteradas de forma que a ordem das operações resulta invertida. Além disso, suponha que a precisão das variáveis tenha sido reduzida (o

modelo de referência utilizava variáveis do tipo `float`, enquanto o modelo de baixo nível utiliza variáveis do tipo `sc_fixed`) e que esta perda de precisão seja da ordem de 0.5.

A Figura 5.3 apresenta os resultados obtidos pela técnica proposta (BVG) e pela implementação de um *scoreboard* que utilize a heurística de associar uma entrada do DUV com a primeira entrada compatível no modelo de referência.

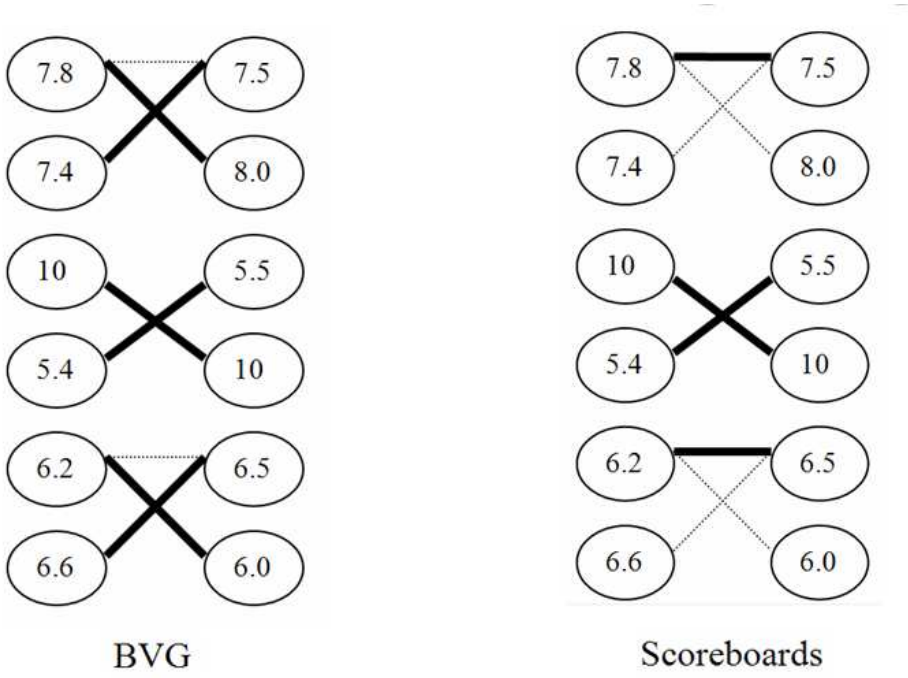

Figura 5.3: Resultados do experimento.

Repare que enquanto o *scoreboard* não consegue associar todos os comportamentos do modelo de referência a algum comportamento do DUV (ou seja o *scoreboard* sinaliza um defeito no modelo) a técnica proposta conseguiu tratar simultaneamente as inversões de ordem e perda de precisão do DUV.

Desta forma, podemos concluir que o *scoreboard*, por necessitar de heurísticas para tomar decisões locais, não consegue prover garantias de falsos positivos e falsos negativos, visto que estas heurísticas limitam o espaço de pesquisa, conforme ilustrado pelo experimento.

Vale fazer alguns comentários sobre o tempo de execução da ferramenta proposta. Conforme já foi demonstrado em [ALB 07], a inclusão de monitores não impacta sobre o tempo de execução de uma plataforma. Além disso, a análise de *logs* de todos os experimentos realizados demandou um tempo muito menor do que a execução da respectiva plataforma. A plataforma MP3, por exemplo, necessita de cerca de 3 horas para finalizar a execução. A análise dos respectivos *logs* demandou menos de 1 segundo.

Capítulo 6

Conclusões e perspectivas

6.1 Conclusões

Esta dissertação formulou um problema de verificação que visa estabelecer a compatibilidade funcional entre duas representações de um componente de um sistema integrado de hardware e software, a partir de um conjunto de monitores e de seus respectivos *logs*.

O problema-alvo foi mapeado de forma que a relação de compatibilidade de valores é capturada por um grafo bipartido. O problema clássico de casamento em grafos bipartidos foi modificado para incluir uma restrição que exclui do espaço de soluções arestas cuja inclusão no casamento violaria a relação de precedência de eventos inerente à especificação.

Um trabalho correlato sobre reflexão computacional [ALB 07] ofereceu a infra-estrutura necessária para a inserção minimamente intrusiva de monitores em representações executáveis de um dado componente do sistema. Por se ter escolhido uma infra-estrutura que viabiliza uma abordagem *white-box*, a técnica proposta nesta dissertação não tem sua observabilidade limitada às entradas e saídas de um componente, ao contrário de um dos métodos convencionais mais frequentemente utilizados [BER 05] [GM 07].

A técnica proposta para a solução do problema formulado não res-

tringe a natureza do componente a ser verificado, o qual poderia ser um módulo de software ou uma representação funcional executável de um módulo de hardware. Assim, a técnica é adequada à etapa de Verificação Pós-Particionamento, onde tanto módulos de hardware como de software precisam ser verificados em relação a uma especificação do sistema. A técnica também se aplica à etapa de Verificação da Implementação, desde que estejam disponíveis representações executáveis do componente, antes e depois de sua implementação (por exemplo, uma representação comportamental de um componente de hardware e sua representação RTL, ambas escritas em SytemC).

A formulação do problema-alvo foi deliberadamente geral para incluir o desafiante cenário de verificação [GM 07] em que um DUV pode ou não executar comportamentos fora da ordem estabelecida no modelo de referência adotado. Assim, por um lado, para o caso particularíssimo (mas de aplicação prática improvável) em que não exista qualquer relação de precedência especificada entre eventos (ou seja, todos os eventos são concorrentes), o algoritmo proposto poderia ser substituído por um algoritmo clássico de casamento em grafos bipartidos. Por outro lado, para o caso mais realista em que a ordem especificada de alguns eventos é mandatória, mas a ordem de outros eventos não precise ser preservada pelo DUV, foram apresentados algoritmos que garantem, simultaneamente, a compatibilidade entre valores e a causalidade entre eventos.

A formalização do problema e de propriedades de suas soluções permitiu que fossem obtidas garantias formais para a verificação. Dentro dos limites de amostragem de um dado monitor, a técnica proposta:

- *Nunca induz falsos negativos* (um DUV contendo defeito que deixou algum traço nos *logs* nunca passa na verificação);
- *Nunca induz falsos positivos* quando a relação entre valores a verificar é uma relação de equivalência (um DUV sem defeitos sempre passa na verificação quando a relação especificada é de equivalência).

Como a indução de falsos positivos é mais tolerável do que a de

falsos negativos, para a situação em que a relação entre valores é de mera compatibilidade, o esforço computacional para eliminar os falsos positivos não se justificaria. Como o algoritmo proposto assume uma ordem arbitrária de percurso dos vértices do grafo, poderiam ser elaboradas heurísticas para induzir uma ordem de percurso que visasse minimizar a ocorrência de falsos positivos, sem limitar o espaço de pesquisa.

Além da validação formal, foi realizada também a validação experimental da técnica proposta quando aplicada a quatro componentes extraídos de aplicações bem conhecidas (criptografia, multi-mídia e escalonamento dinâmico), submetidos a diferentes cenários de precedência entre eventos (não ordenados, totalmente ordenados, parcialmente ordenados). Através da injeção de falhas nos componentes, comprovou-se a habilidade da técnica proposta em atestar a presença e a ausência de defeitos, além de sua eficácia em contornar as dificuldades impostas pela não-preservação da ordem de comportamentos no DUV e a introdução de artefatos de implementação.

A otimização dos algoritmos propostos visando diminuir sua complexidade média e visando potencializar a exploração de paralelismo ao nível de threads em estações de trabalho com vários núcleos de processamento, permitiu a obtenção dos seguintes indicadores experimentais de eficiência:

- O tempo de verificação médio cresce linearmente com o número de valores compatíveis nos *logs* (arestas do grafo);

- Ao executar a análise em uma estação de trabalho contemporânea, para o caso extremo reportado na Figura 5.1 onde os grafos bipartidos são completos, os eventos estão totalmente ordenados e nenhum defeito é encontrado, os resultados experimentais indicam que o tempo de verificação normalizado em relação ao número de entradas de log (runtime / |V|) é de 11ms/vértice e cresce na razão média de 5,8 quando o número de entradas cresce na razão de 2. O tempo de verificação normalizado em relação ao número de valores compatíveis (runtime / |E|) é de 0,7ms/aresta e cresce na razão média de 2,8 quando o número de valores compatíveis cresce na razão de 4.

Esses valores podem ser interpretados como tempos de verificação de pior caso, uma vez que é de se esperar grafos mais esparsos no caso prático típico. Por exemplo, o BVG associado ao monitor inserido à saída da unidade de emissão de instruções na Seção 5.2.4 (Tabela 5.5, linha 2) tem 880 vértices e 15200 arestas, enquanto o grafo completo do Cenário 3 da Figura 5.1 possui aproximadamente o mesmo número de areastas (16384) mas apenas 256 vértices. Ora, os tempo de verificação são, respectivamente, 1ms/aresta e 4,3 ms/aresta.

6.2 Perspectivas de trabalhos futuros

Ao longo do desenvolvimento deste trabalho, foram observadas algumas limitações da infra-estrutura de reflexão computacional adotada e dos cenários de experimentação escolhidos, as quais poderiam ser contornadas em trabalhos futuros. Além disso, novas oportunidades de aplicação da técnica proposta foram percebidas. Tais extensões ou generalizações são a seguir discutidas:

- **Extensão do mecanismo de reflexão utilizado:** O mecanismo de reflexão atual [ALB 07], quando utilizado no âmbito da infra-estrutura de modelagem de plataformas adotada [ARC 06], possui uma limitação ao manipular ponteiros. Ela ocorre quando um ponteiro faz referência a uma posição de um componente de memória simulado na plataforma (ou seja, uma representação executável de uma memória embarcada), conforme ilustra a Figura 6.1, onde o assim-chamado *Whitebox IP* representa o mecanismo de reflexão proposto em [ALB 07] para encapsular o DUV de forma a permitir sua monitoração. Para acessar uma posição nessa representação da memória embarcada, o DUV não pode simplesmente de-referenciar o ponteiro (*ptr), porque isso causaria uma falha de segmentação. Para acessar a posição de memória referenciada pelo ponteiro, o DUV precisa utilizar o barramento (através da função `read`, que recebe como parâmetro um ponteiro para um dado). No entanto, o *Whitebox IP* proposto em [ALB 07] não possui uma porta para o barramento (não podendo

assim invocar a função `read` previamente mencionada). Embora seja possível estender o mecanismo de reflexão para que obtenha acesso à representação da memória embarcada (através do barramento), isto ainda não resolveria totalmente o problema, pois o mecanismo de reflexão não tem como decidir se um dado ponteiro faz referência a uma memória externa ao DUV (que deveria ser acessada via barramento) ou se faz referência a uma memória interna (que deveria ser acessada através do operador *). Além disso, a adição de uma porta entre o barramento e o *Whitebox IP* vai contra o princípio de uma abordagem minimamente intrusiva. Assim, a implementação atual da técnica aqui proposta está limitada em sua observabilidade de memória enquanto uma extensão não for viabilizada na infra-estrutura de reflexão computacional: se um monitor for anexado a um ponteiro para um objeto, ele não será capaz de identificar alterações no objeto, mas apenas alterações no valor do ponteiro.

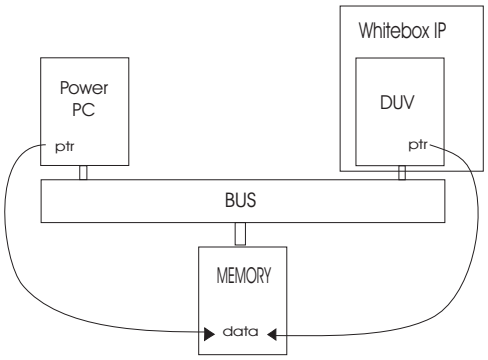

Figura 6.1: IP com um ponteiro para acessar um dado na memória

- **Ampliação do número e da natureza dos experimentos:** A codificação de mais aplicações reais e a injeção de outros tipos de defeitos teve que ser restringida por limitações de tempo e de recursos humanos. Entretanto, uma experimentação mais extensiva poderia ser realizada com casos de verificação extraídos de um conjunto mais amplo de aplicações de sistemas embarcados. Além disso, os defeitos inseridos nos experimentos aqui reportados foram um tanto severos e, portanto, mais fáceis de encontrar. Seria adequado encontrar

cenários de injeção de defeitos mais sutis para avaliar o desempenho da técnica sob condições de maior dificuldade em encontrar defeitos.

- **Utilização do mecanismo de análise de *logs* para avaliar a cobertura:** Os *logs* de um monitor possuem todos os valores observados em um determinado ponto de um componente dentro de um intervalo de monitoração. Assim, esses valores podem ser utilizados para avaliar a cobertura de verificação. Suponha, por exemplo, que um monitor é associado a uma variável que armazena o estado atual de uma máquina de estados finitos. Uma varredura nos registros de *log* deste monitor revelaria todos os estados assumidos pela máquina durante a execução. A proporção entre os estados assumidos e o número total de estados poderia ser utilizada para orientar a escolha de novos estímulos que induzissem estados não alcançados pela máquina. Como produto secundário deste trabalho, toda a infra-estrutura para se contruir um protótipo de uma tal ferramenta de análise de cobertura já está disponível e seria uma extensão natural e complementar ao trabalho aqui reportado.

- **Utilização do mecanismo de *log* para avaliar asserções temporais**

  As informações contidas nos arquivos de *log* podem ser utilizadas para alimentar um mecanismo de asserção. Por exemplo, a cada amostra que a ferramenta recebesse para um determinado monitor ela poderia verificar se o valor obtido esta numa faixa de valores válidos. Além disso, cada elemento de um *log* está associado ao exato instante de tempo t_i em que houve uma transição de valor. O uso destas informações poderia ser utilizada para avaliar relações como: sempre que uma requisição r for registrada num monitor m, então uma resposta a deve ser registrada no monitor n em até x ciclos de relógio. Quando uma violação numa asserção fosse disparada o mecanismo de captura de *logs* poderia, inclusive, paralisar a execução atual (isto é possível através do *IP Whitebox*). A princípio, o único obstáculo para implementação deste mecanismo está na maneira como as entradas de *logs* estão armazenadas. Atualmente as entradas estão indexadas pelos monitores aos quais estão associadas, enquanto que,

para este mecanismo, o ideal seria que as entradas de *log* estivessem indexadas pelo instante em que ocorra mudança de valor. Embora um simples conversor de formato pudesse ser facilmente implementado, o volume de dados de um arquivo de *log* poderia ser demasiadamente grande para tornar proibitivo o tempo necessário para a conversão. Por isso, uma re-implementação seria mais eficiente.

- **Avaliação do tempo de execução em plataformas mais complexas**

  Uma das grandes limitações da técnica proposta em relação as técnicas correlatas reside no tempo computacional necessário para análise de *logs*. Para determinar o impacto desta limitação, o ideal seria aplicar a técnica proposta sobre plataformas mais complexas. Existem algumas maneiras de reduzir o tempo necessário para análise dos *logs*: O elevado paralelismo do algoritmo proposto (em especial a análise de arestas), tende a reduzir o impacto da complexidade; outro aspecto relevante é que em alguns casos pode ser possível particionar um *log* grande em vários *logs* pequenos. Por exemplo, um grande *log* de uma plataforma MP3 poderia ser particionado em vários *logs* independentes, onde cada *log* diz respeito a um determinado *frame*.

- **Uso de transactors**

  Transactors [BOM 06] são utilizados para traduzir vetores de teste entre diferentes níveis de abstração. Ora, esta técnica poderia ser utilizada para relaxar a Premissa 5, auxiliando o mapeamento de entidades em diferentes níveis de abstração.

- **Relaxação da aproximação do Teorema de Hall**

  Seria adequado comparar o custo-benefício obtido quando o Teorema de Hall é aplicado integralmente sobre os experimentos realizados ao invés da aproximação proposta no Teorema 5. Esta alteração resultaria num sacrifício do *runtime* em troca de uma maior qualidade de verificação. Os Teoremas apresentados nesta dissertação sugerem que a utilização integral do Teorema de

Hall (Teorema 5) resulta numa técnica que não apresenta falsos positivos nem falsos negativos.

Os diferentes usos dos dados amostrados em aspectos complementares do processo de verificação parecem indicar que a verificação baseada em *logs* é uma alternativa promissora para Verificação Funcional.

Referências Bibliográficas

[ALB 07] ALBERTINI, B. et al. A computational reflection mechanism to support platform debugging in systemc. In: CODES+ISSS '07: 5TH IEEE/ACM INTERNATIONAL CONFERENCE ON HARDWARE/-SOFTWARE CODESIGN AND SYSTEM SYNTHESIS, 2007. **Proceedings...** New York, NY, USA: ACM, 2007. p.81–86.

[ALP 03] ALPHEY, J. et al. **STAR-IP Centric Platforms for SOC**, chapter9. Kluwer Academic Press, 2003.

[ALT 91] ALT, H. et al. Computing a maximum cardinality matching in a bipartite graph in time $o(n^{1.5}\sqrt{\frac{m}{logn}})$. **Inf. Proc. Letters.**, [S.l.], v., 1991.

[ARC 06] ARCHC. **ARP**. Disponível em <http://archc.sourceforge.net/>. Acesso em: january.

[BAI 05] BAILEY, B. **The Functional Verification of Electronic Systems: An Overview from Various Points of View**. International Engineering Consortium, 2005.

[BB 05] BRIAN BAILEY, T. A. **Taxonomies for the Development And Verification of Digital Systems**. Springer, 2005.

[BER 02] BERGAMASCHI, R. A to z of socs. Tutorial presented at SBC Sul Microelectronics School (EMICRO 2002), 2002.

[BER 05] BERGERON, J. et al. **Verification Methodology Manual for SystemVerilog**. Springer; 1 edition (September 28, 2005), 2005.

[BOM 06] BOMBIERI, N.; FUMMI, F.; PRAVADELLI, G. On the evaluation of transactor-based verification for reusing tlm assertions and testbenches at rtl. In: DATE '06: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2006. **Proceedings...** 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006. p.1007–1012.

[BW 03] BRUCE WILE, JOHN C. GOSS, W. R. **Comprehensive Functional Verification: The Complete Industry Cycle**. Morgan Kaufmann, 2003.

[CAR 99] CARLONI, L. P. et al. A methodology for correct-by-construction latency insensitive design. **iccad**, Los Alamitos, CA, USA, v.00, p.309, 1999.

[COR 90] CORMEN, T. H.; LEISERSON, C. E.; RIVEST., R. L. **Introduction to algorithms**. McGraw-Hill, 1990.

[COR 00] CORTéS, L. A.; ELES, P.; PENG, Z. Verification of embedded systems using a petri net based representation. In: ISSS '00: 13TH INTERNATIONAL SYMPOSIUM ON SYSTEM SYNTHESIS, 2000. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2000. p.149–155.

[FEI 73] FEISTEL, H. Cryptography and computer privacy. **Scientific American**, [S.l.], v.228, p.15–23, 1973.

[FEN 97] FENG, X.; HU, A. J. Early cutpoint insertion for high-level software vs. rtl formal combinational equivalence verification. In: DAC '06: 43RD ANNUAL CONFERENCE ON DESIGN AUTOMATION, 1997. **Proceedings...** New York, NY, USA: ACM, 1997. p.1063–1068.

[GHE 05] GHENASSIA, F. **Transaction-Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems**. Springer; 1 edition (November 28, 2005), 2005.

[GM 07] GRANT MARTIN, BRIAN BAILEY, A. P. **ESL Design and Verification: A Prescription for Electronic System Level Methodology (Systems on Silicon)**. Morgan Kaufmann, 2007.

[GUT 01] GUTHAUS, M. R. et al. Mibench: A free, commercially representative embedded benchmark suite. In: WWC '01: WORKLOAD CHARACTERIZATION, 2001. WWC-4. 2001 IEEE INTERNATIONAL WORKSHOP ON, 2001. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2001. p.3–14.

[HAB 06] HABIBI, A. et al. Efficient assertion based verification using tlm. In: DATE '06: CONFERENCE ON DESIGN, AUTOMATION AND TEST IN EUROPE, 2006. **Proceedings...** 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2006. p.106–111.

[HAL 56] HALL, M. An algorithm for distinct representatives. **The American Mathematical Monthly, No. 10**, [S.l.], v.63, p.716–717, 1956.

[HOP 73] HOPCROFT, J. E.; KARP, R. M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. **SIAM Journal on Computing**, [S.l.], v.2, n.4, p.225–231, 1973.

[IBM 08] IBM. **CoreConnect bus architecture**. Disponível em <http://www-03.ibm.com/technology/ges/semiconductor/power/licensing/coreconnect>. Acesso em: 07-Fevereiro.

[INS 08] INSTRUMENTS, T. **OMAP**. Disponível em <http://www.ti.com/>. Acesso em: 07-Fevereiro.

[KAR 81] KARP, R.; SIPSER, M. Maximum matching in sparse random graphs. In: FOUNDATIONS OF COMPUTER SCIENCE, 1981. SFCS '81.

22ND ANNUAL SYMPOSIUM ON, 1981. **Proceedings...** Nashville, TN, USA: [s.n.], 1981. p.364–375.

[KAS 07] KASUYA, A.; TESFAYE, T. Verification methodologies in a tlm-to-rtl design flow. In: DAC '07: 44TH ANNUAL CONFERENCE ON DESIGN AUTOMATION, 2007. **Proceedings...** New York, NY, USA: ACM, 2007. p.199–204.

[MAR 03a] MARTIN, G.; BAILEY, B.; PIZIALI, A. **Writing Testbenches: Functional Verification of HDL Models, 2nd ed.** Springer, 2003.

[MAR 03b] MARTIN, G.; CHENG, H. **Winning the SoC Revolution: Experiences in Real Design**. Kluwer Academic Press, 2003.

[MER 90] MERKLE, R. C. **One Way Hash Functions and DES**. Springer Berlin / Heidelberg, 1990.

[MEY 04] MEYER, A. **Principles of Functional Verification**. Newnes, 2004.

[MG 03] MCGRODDY-GOETZ, K. et al. **SOC - The IBM Microeletronics Approach**, chapter6. Kluwer Academic Press, 2003.

[NXP 08] NXP. **Nexperia**. Disponível em <http://www.nxp.com/>. Acesso em: 07-Fevereiro.

[OSC 08] OSCI. **Open SystemC Initiative**. Disponível em <http://www.systemc.org/home>. Acesso em: 08-Novembro.

[PAT 07] PATTERSON, D. A.; HENNESSY, J. L. **Computer Organization and Design: The Hardware/Software Interface. Third Edition**. Morgan Kaufmann; 3 edition (June 1, 2007), 2007.

[SCH 94] SCHNEIER, B. **Fast Software Encryption**, chapterDescription of a new variable-length key, 64-bit block cipher (Blowfish). Springer Berlin / Heidelberg, 1994.

[SI 04] SASAN IMAN, S. J. **The e-Hardware Verification Language**. Springer, 2004.

[SV 01] SANGIOVANNI-VINCENTELLI, A.; MARTIN, G. Platform-based design and software design methodology for embedded systems. **IEEE Design Test of Computers**, [S.l.], v.18, n.6, p.23–33, Nov/Dec 2001.

[SWA 06] SWAN, S. Systemc transaction level models and rtl verification. In: DAC '06: 43RD ANNUAL CONFERENCE ON DESIGN AUTOMATION, 2006. **Proceedings...** New York, NY, USA: ACM, 2006. p.90–92.

[VAR 07] VARDI, M. Y. Formal techniques for systemc verification. In: DAC '07: 44TH ANNUAL CONFERENCE ON DESIGN AUTOMATION, 2007. **Proceedings...** New York, NY, USA: ACM, 2007. p.188–192.

[YIM 97] YIM, J.-S. et al. Verification methodology of compatible microprocessors. In: DESIGN AUTOMATION CONFERENCE 1997. ASP-DAC '97. ASIA AND SOUTH PACIFIC, 1997. **Proceedings...** Chiba, Japan: [s.n.], 1997. p.173–180.