FABRÍCIO VIVAS ANDRADE

CONTRIBUIÇÕES PARA O PROBLEMA DE VERIFICAÇÃO DE EQUIVALÊNCIA COMBINACIONAL

Belo Horizonte

22 de agosto de 2008

FABRÍCIO VIVAS ANDRADE

CONTRIBUIÇÕES PARA O PROBLEMA DE VERIFICAÇÃO DE EQUIVALÊNCIA COMBINACIONAL

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Belo Horizonte

22 de agosto de 2008

Resumo

O objetivo desse trabalho é apresentar duas contribuições importantes para o problema de Verificação de Equivalência Combinacional (CEC, do Inglês, *Combinational Equivalence Checking*). A primeira contribuição importante é uma técnica de pré-processamento que deriva informações redundantes dos dois circuitos sob CEC de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade (SAT) para a prova de equivalência entre ambos circuitos. Através dessa técnica, implementada em uma ferramenta denominada Vimplic, é possível superar em desempenho as principais ferramentas do estado da arte de CEC baseado em SAT. É importante ressaltar que a técnica de pré-processamento proposta é formalizada de modo a garantir a exatidão das implicações derivadas e assegurar que a mesma não produz falsos negativos e nem falsos positivos em relação à equivalência dos circuitos sob CEC. Além de detalhes de implementação da Vimplic, o presente trabalho também apresenta uma revisão bibliográfica completa das técnicas de CEC e, principalmente, das técnicas de pré-processamento para SAT. Finalmente, através da aplicação da ferramenta Vimplic, é possível estabelecer relações importantes entre o presente trabalho e os trabalhos na área de Satisfabilidade através do estudo de redundância em fórmulas em CNF.

A segunda contribuição importante proposta é uma ferramenta para geração de circuitos, a BenCGen, que tem como principal objetivo a produção de circuitos para benchmarks. Essa ferramenta é capaz de gerar 24 tipos de circuitos diferentes com tamanhos parametrizados. Variando-se do menor para o maior tamanho de cada circuito, mais de 1.000.000 circuitos podem ser gerados. Tal ferramenta vem suprir uma grande demanda de novos benchmarks para CEC e para outras áreas de Verificação Formal. É importante ressaltar que a maior parte dos circuitos gerados pela ferramenta foram provados corretos. Além disso, uma revisão bibliográfica dos principais benchmarks para a área de Verificação Formal é mostrada no presente trabalho, na qual são destacados os seus principais benefícios e limitações. Finalmente, um comparativo entre os resolvedores de Satisfabilidade mais eficientes na resolução de instâncias de problemas de CEC é apresentado. O comparativo foi feito por meio de um benchmark produzido pela ferramenta BenCGen e através do mesmo foi possível indicar o resolvedor de SAT mais adequado para os problemas de CEC estudados.

Abstract

The key goal of this work is to provide two major contributions to the Combinational Equivalence Checking (CEC) problem. The first major contribution is a preprocessing technique that derives redundant information of the two circuits under CEC in order to decrease the SAT solver solving time used to prove equivalence between the circuits. Through this technique, which was implemented in a tool called Vimplic, we have been able to dramatically reduce the overall verification time of several circuits outperforming the state-of-the-art techniques for CEC. This technique has been formalized in order to assure correctness of the derived implications and also to guarantee that it does not produce results with false-positives or false-negatives according to the equivalence between the circuits under CEC. Besides presenting Vimplic's implementation details, this work also describes a complete bibliographic review of the CEC techniques, specially of the SAT-based preprocessing techniques. Finally, by means of Vimplic tool, relations among the present work and other works on Satisfiability has been established with respect to the study of redundancy in Conjunctive Normal Form (CNF) formulas.

The second major contribution presents a digital circuit generation tool (BenCGen) for benchmarks. This tool can be used to generate 24 very popular types of circuits with parameterized size. More than 1,000,000 different designs may be produced using this tool, ranging from the smallest to the largest size of each circuit. Since there is a growing need for new benchmark circuits, BenCGen can supply a wide range of circuit to supply this demand. Correctness is a significant feature of the circuits generated by this tool. In addition, a complete bibliographic review of the most popular benchmarks for Formal Verification is presented. Finally, a comparison among the most efficient SAT solvers is performed and presented using a large benchmark of CEC instance. The selected benchmark was produced by BenCGen and the results of this comparison point out the most appropriate SAT solver for CEC instances.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xi**

1 Introdução **1**
1.1 Objetivo e Contextualização 1
1.2 Breve Histórico de Abordagens para Verificação de Equivalência de Circuitos Combinacionais 3
1.3 Breve Histório dos Benchmarks para Verificação Formal 5
1.4 Motivação 6
1.5 Contribuições 7
1.6 Organização 7

2 Fundamentação Teórica **8**
2.1 Introdução 8
2.2 Satisfabilidade (SAT) 8
2.2.1 Introdução 8
2.2.2 Resolvedores de SAT 9
2.2.3 Estrutura Básica de um Resolvedor de SAT Atual Baseado em DPLL 9
2.2.4 Heurísticas de Dedução, Diagnóstico e Decisão 12
2.3 Aprendizado em Circuitos Combinacionais 18
2.3.1 Introdução 18
2.3.2 Definições Básicas 18
2.4 Grafo de Implicações 21

3 Verificação de Equivalência de Circuitos Combinacionais **24**
3.1 Introdução 24
3.2 Classificações para CEC 25
3.2.1 Motores de Resolução para CEC 25
3.2.2 Características Internas dos Circuitos sob CEC 26
3.2.3 Particionamento dos Circuitos para CEC 27
3.3 Heurísticas de Particionamento e Cortes 28
3.4 CEC Utilizando ATPG 30
3.5 CEC Utilizando BDD e BMD 31
3.6 CEC Utilizando RL 33
3.7 CEC Utilizando Resolvedor de SAT 34
3.8 CEC Utilizando Motores Híbridos 36

4 Pré-Processamento para Verificação de Equivalência Baseado em SAT 37
4.1 Introdução . . . 37
4.2 Trabalhos Relacionados . . . 38
4.3 A Nova Técnica de Pré-Processamento Proposta . . . 41
4.4 Análise da Técnica de Pré-Processamento Proposta . . . 44
4.4.1 Introdução . . . 44
4.4.2 Determinação de Quais Cláusulas Derivar e Armazenar . . . 44
4.4.3 Métodos de Derivação de Implicações Simples, Diretas e Indiretas 45
4.4.4 Grafo de Implicações Completo ou Parcial . . . 46
4.4.5 Considerações Sobre a Forma de Derivação de Implicações Implementada . . . 46
4.5 Formalização da Técnica de Pré-processamento de Circuitos . . . 47
4.5.1 Proposição Geral . . . 47
4.5.2 Garantindo Exatidão das Implicações Simples . . . 48
4.5.3 Garantindo Exatidão das Implicações Diretas . . . 49
4.5.4 Garantindo Exatidão das Implicações Indiretas . . . 50
4.5.5 Garantindo Exatidão da Implementação da Vimplic . . . 50
4.6 Resultados . . . 52

5 Ferramenta para Geração de Circuitos para Benchmarks - BenCGen 53
5.1 Introdução . . . 53
5.2 Análise dos Benchmarks Mais Populares . . . 55
5.3 Geração Sintética de Benchmarks - BenCGen . . . 58
5.3.1 Introdução . . . 58
5.3.2 Detalhes de Implementação . . . 60
5.3.3 Provando a Exatidão dos Circuitos Gerados . . . 61
5.4 Resultados de CEC Usando a BenCGen . . . 62
5.5 Análise dos Resultados . . . 65

6 Resultados 71
6.1 Introdução . . . 71
6.2 Características dos Experimentos . . . 71
6.3 Seleção de Ferramentas do Estado da Arte de CEC Baseado em SAT . 71
6.4 Resultados Utilizando o Benchmark ISCAS 85 . . . 72
6.5 Resultados Utilizando um Benchmark Produzido pela BenCGen . . . 74

7 Conclusões e Trabalhos Futuros 86
7.1 Conclusões . . . 86
7.2 Trabalhos Futuros . . . 87

A Lista de Siglas 90

Lista de Figuras

2.1 Exemplo de um grafo de implicações para análise de conflitos 17
2.2 Exemplo de uma porta não-justificada 19
2.3 Exemplo de implicações simples e diretas 20
2.4 Exemplo de implicações indiretas e estendidas 21
2.5 Circuito que origina o grafo de implicações da Figura 2.6 22
2.6 Grafo de implicações referente ao circuito da Figura 2.5 23

3.1 Exemplo de Fan-In Transitivo 28
3.2 Exemplo de cortes em circuitos sob CEC 29
3.3 Exemplo de um circuito *miter* para dois circuitos com sete saídas primárias 31

Lista de Tabelas

5.1 Características dos circuitos que podem ser gerados pela ferramenta BenCGen 59
5.2 Resultados para CEC de duas cópias de um mesmo multiplicador *Carry LookAhead* 63
5.3 Resultados para CEC de duas cópias de um mesmo multiplicador *Wallace* 64
5.4 Resultados para CEC de duas cópias de um mesmo multiplicador *Array* 65
5.5 Resultados para CEC de duas cópias de um mesmo multiplicador *Dadda Tree* 66
5.6 Resultados para CEC de duas cópias de um mesmo divisor *Restoring* . 68
5.7 Resultados para CEC de duas cópias de um mesmo divisor *Non Restoring* 69
5.8 Resultados para CEC de duas cópias de um mesmo somador *Ripple Carry* 70

6.1 Resultados para Instâncias de CEC derivadas do Benchmark ISCAS 85 73
6.2 Resultados para Instâncias de CEC de Somadores *Carry Save* de Tamanhos Variados 76
6.3 Resultados para Instâncias de CEC de Somadores *Carry Skip* de Tamanhos Variados 76
6.4 Resultados para Instâncias de CEC de Somadores *Carry Select* de Tamanhos Variados 77
6.5 Resultados para Instâncias de CEC de Somadores *Block Carry LookAhead* de Tamanhos Variados 78
6.6 Resultados para Instâncias de CEC de Somadores *Ripple Carry* de Tamanhos Variados 79
6.7 Resultados para Instâncias de CEC de Multiplicadores *Wallace Tree* de Tamanhos Variados 79
6.8 Resultados para Instâncias de CEC de Multiplicadores *Reduced Tree* de Tamanhos Variados 80
6.9 Resultados para Instâncias de CEC de Multiplicadores *Carry LookAhead* de Tamanhos Variados 81
6.10 Resultados para Instâncias de CEC de Multiplicadores *Array* de Tamanhos Variados 82
6.11 Resultados para Instâncias de CEC de Multiplicadores *Dadda Tree* de Tamanhos Variados 83
6.12 Resultados para Instâncias de CEC de Divisores *Restoring* de Tamanhos Variados 83
6.13 Acréscimo (+) ou Decréscimo (-) no número de cláusulas após o pré-processamento 84

Lista de Algoritmos

1 Algoritmo genérico de um resolvedor de SAT atual baseado em DPLL . 11
2 Algoritmo de pré-processamento implementado na Vimplic 43

Capítulo 1

Introdução

1.1 Objetivo e Contextualização

O objetivo desse trabalho é apresentar duas importantes contribuições para o problema de Verificação de Equivalência de Circuitos Combinacionais[1] (CEC, do Inglês, *Combinational Equivalence Checking*). A primeira contribuição é uma técnica de pré-processamento que deriva informações redundantes dos dois circuitos sob CEC de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade (SAT) para a prova de equivalência entre ambos circuitos. Através dessa técnica, é possível superar em desempenho as principais técnicas do estado da arte de CEC baseado em SAT. A segunda contribuição é uma ferramenta para geração de circuitos que tem como principal objetivo a produção de circuitos para benchmarks. Essa ferramenta é capaz de gerar 24 tipos de circuitos diferentes com tamanhos parametrizados. Variando-se do menor para o maior tamanho de cada circuito, mais de 1.000.000 circuitos podem ser gerados. Tal ferramenta vem suprir uma grande demanda de novos benchmarks para CEC e para outras áreas de Verificação Formal.

A CEC é uma das técnicas mais bem sucedidas e mais utilizadas no campo da Verificação Formal de circuitos integrados. A Verificação de Equivalência consiste em provar equivalência funcional entre dois circuitos quaisquer, isto é, dada qualquer combinação de entrada para ambos, as saídas dos mesmos devem ser idênticas.

Circuitos combinacionais representam uma das duas principais classes de circuitos digitais existentes. Caracterizam-se por não possuírem nenhum elemento de memória e, por isso, as suas saídas são sempre dadas somente em função das entradas. Esses circuitos são utilizados quando é necessário aplicar uma transformação de um conjunto de informações ou sinais fornecidos na entrada para um conjunto de informações ou sinais desejados na saída. Além dos circuitos combinacionais, a outra classe principal de circuitos digitais é chamada de circuitos seqüenciais. A saída desses circuitos depende

[1] O termo Verificação de Equivalência de Circuitos Combinacionais é equivalente a Verificação de Equivalência Combinacional e, por isso, ambos termos serão utilizados ao longo do presente trabalho.

tanto da entrada fornecida quanto do estado interno do circuito. A sua construção é feita através do agrupamento de elementos de memória e de circuitos combinacionais.

Resolvedores de SAT são programas que implementam algoritmos que buscam assinalamentos para as variáveis de uma dada fórmula proposicional, com o intuito de torná-la verdadeira ou de provar que tal assinalamento inexiste. Atualmente, muitos resolvedores de SAT são empregados para resolver instâncias de problemas de CEC, porque esses problemas podem ser facilmente transformados em fórmulas proposicionais.

A dificuldade no problema de CEC advém do fato da crescente demanda de projetos de circuitos digitais complexos com centenas de milhares de portas lógicas em curtos espaços de tempo e, também, da inerente complexidade teórica do problema. A CEC é um problema co-NP-difícil [MM04], no entanto, uma boa parte das instâncias derivadas de circuitos integrados é mais tratável.

O problema de CEC acontece em diferentes estágios do projeto de um circuito integrado. A partir de uma especificação em Nível de Transferência de Registros (RTL, do Inglês, *Register Transfer Level*), um processo de síntese é feito gerando um conjunto de portas lógicas que representam o circuito sintetizado. Em seguida, o circuito sintetizado pode sofrer otimizações e mapeamento tecnológico, gerando um circuito sintetizado e otimizado. Finalmente, o circuito sintetizado e otimizado é transformado em um circuito físico. A Verificação de Equivalência é executada em cada um dos estágios para garantir que o circuito antes e o depois da transformação são funcionalmente equivalentes; atualmente ela é mais utilizada para garantir exatidão entre um circuito e a sua cópia após a intervenção manual.

Para validar e avaliar as novas técnicas e metodologias de CEC, benchmarks de circuitos são extremamente importantes na área de Automação de Projetos Eletrônicos (EDA, do Inglês, *Electronic Design Automation*). Entretanto, como a escala de integração dos circuitos integrados tem aumentado significativamente ao longo dos anos, os benchmarks de circuitos existentes estão se tornando cada vez mais inadequados para tal finalidade. Por esse motivo, a busca por novos benchmarks é essencial para o desenvolvimento da indústria de circuitos integrados.

1.2 Breve Histórico de Abordagens para Verificação de Equivalência de Circuitos Combinacionais[2]

Diversas metodologias bastante eficientes para lidar com o problema de CEC já foram propostas. Dentre estas, destaca-se a utilização das técnicas de Geração Automática de Padrões de Teste (ATPG, do Inglês *Automatic Test Pattern Generation*), de Diagrama de Decisão Binária (BDD[3], *Binary Decision Diagram*), de resolvedores de SAT, de Aprendizado Recursivo (RL, do Inglês *Recursive Learning*) e de técnicas híbridas como BDD em conjunto com resolvedores de SAT, Aprendizado Recursivo com resolvedores de SAT, dentre outros.

A utilização de ATPG para o problema de CEC ofereceu resultados promissores inicialmente. O primeiro passo dessa técnica consiste na construção de um *miter* (Seção 3.4) a partir dos dois circuitos a serem provados equivalentes. Para qualquer combinação de entradas, se a saída do *miter* indicar o valor 1, significa que os circuitos não são equivalentes. Assim, basta utilizar o modelo de falhas *stuck-at* (Seção 3.4) de ATPG, isto é, procurar por falhas do tipo *stuck-at-0*, pois essas apresentam os contra-exemplos que provam que os circuitos não são equivalentes.

Resolver instâncias do problema de CEC, utilizando ATPG, funciona bem apenas se o tamanho do circuito *miter* é pequeno, o que não acontece na prática. Se os circuitos que compõem o *miter* forem similares, ainda é possível particionar o problema em outros menores (Seção 3.3). Caso contrário, essa técnica torna-se pouco viável.

Paralelamente ao desenvolvimento de metodologias baseadas em ATPG, metodologias baseadas em Aprendizado Recursivo eram exploradas. O RL, como originalmente proposto, tem como objetivo derivar implicações a partir da estrutura topológica do circuito *miter*. O procedimento consiste em assinalar valores às saídas, ou às entradas de portas lógicas e propagar as implicações geradas (Seção 3.6). A partir de um conjunto dessas implicações, é possível identificar similaridades internas entre os dois circuitos sob CEC e reduzir o número de implicações necessárias para provar a equivalência entre ambos. No entanto, caso similaridades internas não sejam encontradas, o número de implicações cresce exponencialmente inviabilizando a utilização da técnica de RL.

As metodologias que utilizam BDD para solucionar instâncias do problema de CEC geralmente executam dois passos principais. Primeiramente, constroem-se os BDDs para ambos os circuitos a serem provados equivalentes. Devido à propriedade de canon-

[2]Esta seção tem por objetivo fazer um breve histórico das abordagens utilizadas para resolver instâncias do problema de CEC. Nesta seção, são apresentados conceitos comuns na literatura de Verificação Formal de circuitos integrados. Para os não familiarizados com essa literatura, a cada conceito introduzido, a seção onde o mesmo está definido é informada entre parênteses. Além disso, o foco desta seção é apenas em abordagens puras, isto é, abordagens que utilizam apenas uma metodologia.

[3]Na realidade, o termo mais correto seria ROBDD (do Inglês, *Reduced Ordered Binary Decision Diagram*). Entretanto, em CEC, o termo BDD já é utilizado como sinônimo de ROBDD, pois, na imensa maioria dos casos, o BDD sempre acaba sendo ordenado e reduzido.

icidade do BDD, se um BDD de um circuito qualquer for isomórfico a um BDD de outro circuito, os circuitos são equivalentes. Infelizmente, a construção do BDD para o circuito inteiro não é possível em grande parte dos casos devido à necessidade de espaço de memória exponencial.

Assim como em ATPG, a abordagem utilizada para as metodologias com a técnica BDD foi o particionamento do circuito *miter*, principalmente, através da Verificação de Equivalência de saídas separadamente, ao invés de se verificar todas as saídas simultaneamente. Todavia, em circuitos Estruturalmente Dependentes (Seção 3.2.2), o particionamento por saídas separadamente não resulta em nenhuma vantagem.

A partir daí, foi necessário evoluir para as técnicas de cortes que efetuam partições dentro de partições do circuito (Seção 3.3). Apesar de as técnicas que usam BDDs serem muito boas e já serem utilizadas na indústria de circuitos integrados por mais de 15 anos, algumas limitações como o problema dos *falsos negativos* (Seção 3.5) e a impossibilidade de lidar com circuitos dissimilares motivam o estudo de novas abordagens (Seção 3.5).

Uma das principais metodologias para lidar com essas limitações é a CEC utilizando resolvedores de SAT; tal metodologia segue os seguintes passos. Primeiramente, constrói-se o *miter*. Em seguida, o *miter* é transformado em cláusulas em Forma Normal Conjuntiva (CNF, do Inglês, *Conjunctive Normal Form*), já que essa é a entrada padrão dos resolvedores de SAT mais populares. A partir daí, basta criar uma propriedade que obrigue a saída do miter ser 1. Se o resolvedor encontrar algum assinalamento para as variáveis da fórmula com a propriedade, esse será um contra-exemplo que prova que os circuitos não são equivalentes (Seção 3.7).

Algumas dificuldades surgem quando resolvedores de SAT são utilizados no problema de CEC. A principal delas é que a estrutura topológica do circuito é perdida durante a transformação para CNF. Assim, dificilmente, similaridades internas poderão ser encontradas como nas outras técnicas. No entanto, grandes vantagens advogam para a utilização de resolvedores de SAT, principalmente a crescente eficiência dos resolvedores e a capacidade de lidar tanto com circuitos similares quanto com dissimilares.

Para avaliar o desempenho das várias técnicas existentes para cada uma das metodologias apresentadas anteriormente, benchmarks de circuitos são extremamente necessários. Dessa forma, a seção seguinte apresenta um breve histórico dos benchmarks mais populares em CEC e em outras áreas de Verificação Formal.

1.3 Breve Histório dos Benchmarks para Verificação Formal

Existem duas principais classes nas quais os benchmarks podem ser divididos de acordo com os tipos de circuitos que os mesmos possuem. Por um lado, existem os benchmarks industriais, e, por outro lado, existem os benchmarks sintéticos. Os benchmarks da primeira classe foram a escolha mais popular durante as décadas de 80 e 90. Os benchmarks mais utilizados em Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais, e em Geração Automática de Padrões de Teste são denominados benchmarks ISCAS (do Inglês, *IEEE International Symposium on Circuits and Systems*). Existem dois tipos de benchmarks ISCAS: o primeiro é formado por circuitos combinacionais (ISCAS 85 [BF85]) e o segundo é formado por circuitos seqüenciais (ISCAS [FBK89]).

Recentemente, outros dois importantes benchmarks foram disponibilizados: benchmarks ITC 99 [Dav99] (do Inglês, *International Test Conference*) e os benchmarks Velev [Vel04]. O primeiro desses é freqüentemente utilizado em ATPG e CEC, enquanto o segundo é usado principalmente em verificação formal utilizando resolvedores de SAT.

Embora os benchmarks industriais relacionados anteriormente tenham sido a escolha predominante em muitas áreas de EDA, os mesmos são muito limitados devido a três aspectos principais. O primeiro aspecto, e o mais importante, é que as empresas de projeto de circuitos integrados não fornecem projetos (circuitos) atualizados devido ao seu alto valor de Propriedade Intelectual agregado. Por isso, a imensa maioria dos benchmarks industriais disponíveis possui circuitos desatualizados tanto em complexidade quanto em funcionalidade, mesmo aqueles disponibilizados há poucos anos. O segundo aspecto é que apenas um número muito limitado de circuitos torna-se público, o que também é justificado pelo alto valor de Propriedade Intelectual dos mesmos. O terceiro aspecto é que, freqüentemente, apenas circuitos com formatos, funcionalidades e tamanhos bem específicos são fornecidos.

Com o objetivo de superar as limitações dos benchmarks industriais, benchmarks sintéticos têm sido propostos ao longo da última década. Os benchmarks sintéticos são formados por coleções de circuitos gerados por uma ferramenta, de acordo com as características dos circuitos desejados. Uma das primeiras abordagens de sucesso na geração desses benchmarks foi utilizada na área de análise de roteamento em FPGAs (do Inglês, *Field Programmable Gate Array*) [DD96]. Em seguida, vários outros benchmarks sintéticos foram propostos, tais como [MHC02] [PLM99] [VSC02].

O grande problema em se utilizar apenas benchmarks sintéticos é que quase todas as ferramentas de geração desses benchmarks produzem circuitos aleatórios, o que resulta em circuitos que são muito diferentes dos utilizados na indústria de semicondu-

tores. Dessa forma, dependendo do benchmark produzido pela ferramenta de geração, a avaliação de novas técnicas com os mesmos pode levar a conclusões errôneas.

Uma solução intermediária é geração de benchmarks sintéticos de circuitos que são freqüentemente utilizados na indústria de semicondutores como multiplicadores, divisores, multiplexadores, decodificadores, dentre outros. Os grandes benefícios dessa solução é que os circuitos gerados não são tão artificiais quanto aqueles utilizados em benchmarks sintéticos com circuitos aleatórios, e não tão específicos, limitados e, quase sempre, desatualizados como aqueles fornecidos pelos benchmarks industriais. Por esse motivo, o presente trabalho propõe uma alternativa para a geração de benchmarks baseando-se nessa solução intermediária.

1.4 Motivação

Problemas de verificação de circuitos integrados, como o problema de CEC, são muito freqüentes nos dias de hoje, principalmente, porque houve um crescimento acelerado na utilização de pequenos dispositivos de computação embutida nas últimas duas décadas. Segundo Mishra e Dutt [MD04], dois fatores foram os principais motivadores para o aumento: os avanços tecnológicos e a crescente demanda por dispositivos eletrônicos.

Em relação ao primeiro desses fatores, de acordo com a Lei de Moore [Moo65], existe um crescimento exponencial do número de elementos em um circuito integrado. Assim, é de se esperar que o esforço para se projetar e, principalmente, verificar esses circuitos também cresça exponencialmente. Para o segundo fator, a necessidade de se colocar um produto novo no mercado, em períodos de tempo cada vez menores, leva freqüentemente ao lançamento de produtos com defeitos de projeto.

Com o intuito de se evitar defeitos como esses, que geram enormes prejuízos para as empresas desenvolvedoras, a tarefa de verificação tornou-se a parte mais importante e a que mais demanda tempo em um projeto de circuito integrado. Nos dias de hoje, esta tarefa já representa de 50% a 80% do tempo total de desenvolvimento do circuito integrado [FKL04] [Dre04] [BBN+04].

Apesar de toda a dedicação à tarefa de verificação, existem vários projetos de circuitos integrados em que os erros foram apenas detectados após a entrada do produto no mercado. Um dos casos mais famosos é o defeito na unidade de divisão de ponto flutuante do Pentium [Bei95]. O acontecimento obrigou a Intel a gastar aproximadamente 475 milhões de dólares na substituição dos processadores defeituosos.

Ainda sobre os projetos da Intel, segundo Schubert [Sch03], os defeitos lógicos encontrados antes de fazer o primeiro circuito integrado para o Pentium foram de 800, para o Pentium Pro foram de 2240, para o Pentium IV foram de 7855. É uma tendência de crescimento de 300% a 400% no número de defeitos em cada nova geração de processadores.

Dada a relevância do tema, o presente trabalho propõe duas importantes contribuições para um dos principais problemas na área de verificação formal, a Verificação de Equivalência de Circuitos Combinacionais.

1.5 Contribuições

De forma mais ampla, as principais contribuições[4] do presente trabalho são as seguintes:

- Uma técnica de pré-processamento para derivar informações redundantes dos dois circuitos sob CEC de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade (SAT) para a prova de equivalência entre ambos circuitos. Tal técnica, implementada em uma ferramenta, é capaz de superar em desempenho as principais técnicas do estado da arte de CEC baseado em SAT.

- Uma ferramenta para geração de circuitos que tem como principal objetivo a produção de circuitos para benchmarks. Essa ferramenta é capaz de gerar 24 tipos de circuitos diferentes com tamanhos parametrizados. Variando-se do menor ao maior tamanho de cada circuito, mais de 1.000.000 podem ser produzidos.

1.6 Organização

O restante do presente trabalho está organizado da seguinte maneira. O Capítulo 2 fornece toda a fundamentação teórica que dá suporte ao presente trabalho e é dividido em três partes: Resolvedores de SAT, Aprendizado em Circuitos Combinacionais e Grafo de Implicações. O Capítulo 3 apresenta o problema de Verificação de Equivalência de Circuitos Combinacionais, bem como uma explicação detalhada das principais metodologias e técnicas propostas para a resolução do problema. O Capítulo 4 apresenta a técnica de pré-processamento para derivação de informações redundantes proposta juntamente com os seus trabalhos relacionados. Esse capítulo também apresenta as considerações e os detalhes relativos à implementação da ferramenta que aplica a técnica. O Capítulo 5 apresenta os diversos tipos de benchmarks utilizados em EDA mostrando os benefícios e as limitações dos mesmos. Além disso, descreve a ferramenta de geração de circuitos para benchmarks proposta e conclui estabelecendo um comparativo de desempenho entre quatro importantes resolvedores de SAT. O Capítulo 6 apresenta os resultados obtidos com a ferramenta de pré-processamento comparando-os com os resultados das ferramentas do estado da arte de CEC baseado em SAT. Finalmente, o Capítulo 7 apresenta as conclusões e mostra como o presente trabalho pode ser estendido.

[4]Uma descrição mais detalhada das contribuições aparece no Capítulo 7.

Capítulo 2

Fundamentação Teórica

2.1 Introdução

Este capítulo tem por objetivo apresentar a fundamentação teórica que dá suporte ao presente trabalho. A primeira seção trata do problema de Satisfabilidade (SAT) e, principalmente, do seu resolvedor que é a principal ferramenta utilizada; a segunda seção apresenta os conceitos básicos sobre Aprendizado em Circuitos Combinacionais; e, finalmente, a terceira seção apresenta uma estrutura de dados para as ferramentas de derivação de implicações, o grafo de implicações.

2.2 Satisfabilidade (SAT)

2.2.1 Introdução

Satisfabilidade é um problema que aparece em diversos contextos na indústria de circuitos integrados como na Geração Automática de Padrões de Teste, no cálculo de atrasos em circuitos, na Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais, dentre outros. Além das aplicações na indústria de circuitos integrados, esse problema é base para a fundamentação teórica de um dos problemas em aberto mais importantes na área de Ciência da Computação: $P = NP?$. De acordo com o Teorema de Cook [Coo71], o problema de SAT está em P se e somente se $P = NP$.

Uma definição mais formal para o problema de SAT é a seguinte:

Definição 1 *Seja φ uma fórmula proposicional. O problema de SAT consiste em determinar se existe algum assinalamento para a fórmula φ que a torne verdadeira, ou provar que tal assinalamento inexiste.*

Para lidar com o problema, diversos resolvedores de SAT já foram propostos; os mais recentes são zChaff [MMZ+01], Berkmin561 [GN02], Minisat 2 [ES03], Siege_v4 [Rya04] e o RSat [PD07]. Resolvedores de SAT são programas que implementam

algoritmos e heurísticas para solucionar instâncias do problema de SAT. A maior parte dos resolvedores recebe a fórmula φ na Forma Normal Conjuntiva (CNF, do Inglês, *Conjunctive Normal Form*). Uma definição para CNF é a seguinte:

Definição 2 *Uma fórmula φ com n variáveis $x_1, x_2, \ldots, x_n$, em que $x_i \in \{0,1\}$ para $i \in \{1,2,\ldots,n\}$, está na Forma Normal Conjuntiva se esta é formada por uma conjunção ($\wedge$) de m cláusulas $\omega_1, \omega_2, \ldots, \omega_m$ em que cada cláusula é formada pela disjunção ($\vee$) de uma ou mais variáveis em sua forma complementada ou não complementada.*

Freqüentemente, a ocorrência da variável, seja em sua forma complementada ou não-complementada, é denominada literal. Um exemplo de fórmula em CNF é o seguinte:

$$\varphi_1 = \underbrace{(\overline{x}_1)}_{\omega_1} \wedge \underbrace{(x_1 \vee \overline{x}_2)}_{\omega_2} \wedge \underbrace{(x_1 \vee x_3)}_{\omega_3} \wedge \underbrace{(x_1 \vee x_2 \vee \overline{x}_3 \vee x_4 \vee x_5)}_{\omega_4} \wedge \underbrace{(x_1 \vee x_2 \vee x_4 \vee \overline{x}_5)}_{\omega_5}$$

Uma abordagem ingênua para resolver o problema de SAT da fórmula φ_1 é a de fazer uma busca no espaço de soluções gerando todas os assinalamentos possíveis, ou seja, para n variáveis, 2^n assinalamentos. No entanto, existem alternativas melhores que são mostradas na seção seguinte.

2.2.2 Resolvedores de SAT

O primeiro resolvedor de SAT conhecido da literatura foi proposto por Davis e Putnam [DP60]. O resolvedor baseia-se em uma operação bastante eficiente na simplificação de expressões Booleanas que atualmente é denominada *Resolução*. No entanto, apesar de a abordagem representar um grande avanço em relação à simulação de todas as atribuições possíveis na resolução de instâncias do problema de SAT, foi a proposta feita por Davis et al. [DLL62] que ofereceu um desempenho substancialmente melhor. A proposta hoje é conhecida por resolvedor de SAT DPLL (Davis, Putnam, Logemann e Loveland). Grande parte dos resolvedores de SAT atuais ainda utiliza os princípios básicos do resolvedor de SAT DPLL como pode ser visto na próxima seção.

2.2.3 Estrutura Básica de um Resolvedor de SAT Atual Baseado em DPLL

Os resolvedores de SAT atuais baseados no DPLL geralmente possuem implementações bastante simples e a sua estrutura básica já foi apresentada na literatura diversas vezes com um grau maior ou menor de detalhes. Uma exposição bastante didática e que aborda as principais funções necessárias para contextualizar o presente trabalho foi proposta por Silva e Sakallah [SS96] e, por isso, será a referência para os próximos parágrafos.

Um algoritmo de resolvedor de SAT atual baseado no DPLL segue a estrutura apresentada no Algoritmo 1. Esse algoritmo é composto por quatro funções principais: **Decide**, **Deduz**, **Diagnostica** e **Apaga**. O algoritmo faz uma busca no espaço de soluções através do assinalamento de variáveis em uma árvore de decisão usando a função **Decide**. A cada decisão em um nível da árvore, a busca é repetida em um nível mais profundo da mesma. No Algoritmo 1, a variável d corresponde ao nível atual de decisão ou nível atual de profundidade da busca na árvore.

Inicialmente, a função **Decide** escolhe um assinalamento e verifica se o mesmo torna a fórmula verdadeira; caso positivo, o algoritmo retorna satisfazível (SAT) e termina. Caso contrário, significa que mais assinalamentos serão necessários já que apenas um assinalamento não foi suficiente para tornar a fórmula verdadeira. Assim, entra-se em um laço em que a função **Deduz** é ativada.

A função **Deduz** possui um papel fundamental na eficiência dos resolvedores de SAT. Ela é responsável por gerar implicações a partir do assinalamento recém-decidido. A função pode retornar um conflito (inconsistência) que significa que, devido às implicações e atribuições já feitas, a fórmula submetida ao resolvedor não pode ser satisfazível por esse caminho da árvore. Sendo esse o caso, a função **Diagnostica** é ativada.

A função **Diagnostica** analisa por que o conflito foi gerado e indica para qual nível de decisão o retrocesso deve ser feito. No algoritmo, o nível do retrocesso é indicado pela variável β. No entanto, antes do retrocesso, é necessário a ativação da função **Apagar** para eliminar as implicações feitas pela função **Deduz** no nível atual, já que existe um conflito.

Se a função **Deduz** não retornar conflito como suposto anteriormente, uma chamada recursiva do procedimento principal é feita, aprofundando em um nível a busca na árvore de decisão.

Para tornar mais claro o funcionamento do resolvedor, suponha que a fórmula φ_1 do exemplo da Seção 2.2.1 seja fornecida como entrada para um resolvedor de SAT atual baseado em DPLL mostrado no Algoritmo 1. O resolvedor seguiria os seguintes passos.

Primeiro, a função **Decide** escolhe um assinalamento com o intuito de tornar a fórmula verdadeira. Supondo uma ordem lexicográfica nas escolhas, a primeira variável decidida é x_1 e, para satisfazer a cláusula ω_1, a atribuição $x_1 = 0$ é necessária. Segundo, através da função **Deduz**, as implicações (ou assinalamentos) $x_2 = 0$ e $x_3 = 1$ são feitas tendo em vista as cláusulas ω_2 e ω_3, respectivamente.

Nesse ponto, todas as implicações já foram feitas, nenhum conflito foi encontrado e uma nova decisão é necessária. Terceiro, como as variáveis x_1, x_2 e x_3 já possuem assinalamentos, a função **Decide** pode então selecionar a atribuição $x_4 = 0$. As implicações do novo assinalamento são $x_5 = 1$ pela cláusula ω_4 e $x_5 = 0$ pela cláusula ω_5. Note que a variável x_5 tem que assumir dois valores distintos para tornar a fórmula φ_1

Algoritmo 1: Algoritmo genérico de um resolvedor de SAT atual baseado em DPLL

```
ResolvedorSAT (d, β) Início
    // Faz um assinalamento em uma variável escolhida
    Se ( Decide (d) == FÓRMULA_VERDADEIRA ) Início
        Retorne SAT;
    Fim
    Enquanto ( VERDADEIRO ) Início
        // Propaga as conseqüências do assinalamento e
        // verifica se houve conflito.
        Se ( Deduz (d) != CONFLITO ) Início
            // Se não houve conflito, chama o procedimento
            // recursivamente para uma nova decisão.
            Se ( ResolvedorSAT (d + 1, β) == SAT ) Início
                // Se a chamada recursiva retornar SAT,
                // retorna SAT desempilhando uma chamada.
                Retorne SAT;
            Fim
            Senão Se ( β != d || d == 0 ) Início
                // Se a chamada recursiva retornar UNSAT e
                // o procedimento não estiver no nível de decisão atual,
                // apaga as decisões e retorna UNSAT.
                Apaga (d);
                Retorne UNSAT;
            Fim
        Fim
        Se ( Diagnostica (d, β) == CONFLITO ) Início
            Retorne UNSAT;
        Fim
    Fim
Fim
```

satisfazível, isso gera o que é denominado conflito. Quarto, devido ao conflito, a função **Diagnostica** é ativada e retorna insatisfazível (UNSAT), saindo do nível de recursividade atual e alcançando a função **Apaga** que retorna para o nível de decisão anterior[1]. Invertendo a decisão feita no nível 1, ou seja, $x_4 = 0$ para $x_4 = 1$, as cláusulas ω_4 e ω_5 são satisfeitas independemente da atribuição de x_5, gerando assim, uma solução S para a fórmula φ_1, $S = \{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 1, x_5 = X\}$[2]. No Algoritmo 1, essa inversão é feita dentro da função **Decide**.

Apesar de grande parte dos resolvedores atuais utilizarem a estrutura mostrada no Algoritmo 1, as heurísticas utilizadas nas funções: **Decide**, **Deduz** e **Diagnostica** de cada um deles são bastante diferentes entre si. As seções seguintes explicam resumidamente cada uma delas, já que a escolha de um resolvedor de SAT específico traz consigo heurísticas que podem ser mais adequadas para classes específicas de problemas como instâncias de fórmulas aleatórias ou instâncias derivadas diretamente de circuitos digitais.

2.2.4 Heurísticas de Dedução, Diagnóstico e Decisão

Heurística de Dedução

A principal característica das heurísticas de dedução é gerar implicações (assinalamentos de variáveis) de modo eficiente, pois se estima que 90% do tempo gasto pelo resolvedor de SAT é consumido durante tal procedimento [MMZ+01]. Desde a proposição inicial do resolvedor de SAT de Davis and Putnam [DP60], uma regra denominada *Regra da Cláusula de Um Literal* (RCUL) (ou ainda *Regra da Clásula Unitária*) é utilizada em praticamente todos os resolvedores de SAT baseados em DPLL. A regra é baseada na seguinte definição:

Definição 3 *Seja φ uma fórmula em CNF formada por m cláusulas $\omega_1, \omega_2, \ldots, \omega_m$. Se φ possui uma cláusula ω_c qualquer, em que $c \in \{1, 2, \ldots m\}$, que é composta por apenas um literal x, então φ pode ser transformada em um fórmula φ', retirando todas as cláusulas que contém o literal x e retirando todos os literais $\overline{x}$ das cláusulas remanescentes.*

O resultado mais importante da definição 3 é o Lema 1, apresentado em [DP60].

Lema 1 *Seja φ' uma fórmula gerada aplicando-se a Regra da Cláusula de Um Literal a partir de uma fórmula φ em CNF. A fórmula φ' é satisfazível se e somente se φ é satisfazível*[3].

[1] Neste exemplo simples, será assumido que o retrocesso é sempre para o nível de decisão anterior. Esse não é o caso real, como está explicado na Seção 2.2.4.

[2] O símbolo X neste contexto significa Não Importa (do Inglês, *Don't Care*) já que a fórmula é satisfazível tanto para $x_5 = 0$ quanto para $x_5 = 1$.

[3] Uma prova deste Lema pode ser encontrada em [MM04].

Em palavras mais simples, a RCUL indica que sempre que existir uma cláusula com apenas um literal, o mesmo deve ser assinalado com o valor que torne a cláusula verdadeira (por exemplo, se o literal é $\overline{x}$, então $x = 0$). Com isto, as cláusulas que contém o literal tornam-se verdadeiras podendo ser retiradas da fórmula original. Além disso, as cláusulas que contém o literal complementado só podem se tornar verdadeiras em conseqüência dos outros literais, assim o mesmo pode ser eliminado.

Apesar de simples, a RCUL possui conseqüências importantes. A primeira é que a fórmula gerada pode ser bem menor do que a original. Além disso, a aplicação da RCUL pode levar à ocorrência de novas cláusulas com apenas um literal, possibilitando novamente a aplicação da RCUL. O processo iterativo de aplicação da RCUL é denominado Propagação de Restrição Booleana (BCP, do Inglês, *Boolean Constraint Propagation*).

Como afirmado anteriormente, uma grande parte dos resolvedores utilizam a RCUL e o BCP. O que existe de diferente entre eles é a maneira de se implementar a RCUL e de detectar cláusulas com apenas um literal. As propostas mais recentes foram de Silva e Sakallah [SS96] que mantêm contadores com o número de literais assinalados em cada cláusula. Caso o contador seja menor em uma unidade em relação ao número total de literais da cláusula, então existe uma cláusula unitária.

A maior limitação dessa proposta é a necessidade da atualização de todos os contadores quando um retrocesso é feito, pois alguns literais assinalados devido a decisões e implicações voltam ao seu estado original, isto é, sem assinalamentos. Além do mais, a cada assinalamento todos os contadores das cláusulas têm que ser analisados para descobrir se os mesmos atingiram a contagem que indica que existe uma cláusula unitária.

Uma proposta mais recente utiliza a técnica denominada Vigilância por Dois Literais (do Inglês, *Two Watched Literals*) [MMZ+01]. Diferentemente da proposta anterior que verifica cada contador das cláusulas em cada assinalamento feito, a técnica propõe observar uma cláusula com n literais somente quando o número de literais assinalados na mesma passar de $n-2$ para $n-1$.

O procedimento funciona da seguinte forma. No início, dois literais quaisquer não assinalados são escolhidos em cada cláusula para serem vigiados. À medida que literais são assinalados devido a decisões ou implicações, o assinalamento pode recair sobre um literal vigiado ou não. Se não for um literal vigiado, então nenhuma operação é necessária. Se for, é necessário escolher outro literal para compor a dupla de literais vigiados. Se não existirem mais literais não assinalados para formar a dupla, significa que existe apenas um literal não assinalado, portanto, o BCP poderá ser aplicado.

Uma grande vantagem dessa técnica é a redução do número de cláusulas inspecionadas a cada atribuição, já que somente as cláusulas que possuem os literais de vigilância assinalados precisam ser manipuladas. Outro benefício a ser destacado é que não existem

contadores que precisam ser atualizados a cada retrocesso.

Heurística de Diagnóstico

A função de diagnóstico é ativada todas as vezes que um conflito ocorre na função de dedução. O objetivo da função de diagnóstico é analisar quais foram as causas (assinalamentos) que levaram ao conflito e, por meio desta análise, tomar uma decisão que possa restringir o espaço de busca.

As propostas iniciais de uma função para tomar decisões sobre a análise dos conflitos eram extremamente simples. Todas as vezes que um conflito acontecia, a última variável decidida (a variável com o maior nível de profundidade na árvore) tinha o seu assinalamento complementado, isto é, se o assinalamento $x = 0$ decidido no nível $d = d_{n+1}$ gerou um conflito, a função de diagnóstico apagava todas as implicações do nível d_{n+1} e retornava no nível d_n em que x não possuía assinalamento e fazia um assinalamento $x = 1$, entrando em um novo nível d_{n+1}. Note que, nesse caso, o retrocesso é sempre feito do nível d_{n+1} para d_n; por isso, é denominado Retrocesso Cronológico.

Silva e Sakallah [SS96] apresentaram uma nova proposta de análise de conflito que aumentava bastante a eficiência em relação a proposta anterior. A nova proposta é baseada em duas técnicas: o Aprendizado Direcionado por Conflito e o Retrocesso Não Cronológico.

O Aprendizado Direcionado por Conflito consiste em adicionar cláusulas redundantes à fórmula original do problema de modo a evitar que o mesmo conflito se repita. Uma das maneiras de se aprender ou gerar uma cláusula de conflito é através da criação de um Grafo de Implicações para análise de conflitos[4] interno ao resolvedor de SAT.

Um Grafo de Implicações para análise de conflitos é um grafo direcionado em que: cada vértice corresponde a um assinalamento; os vértices adjacentes são gerados a partir de implicações (RCUL); os vértices que não possuem vértices adjacentes correspondem a decisões; e um vértice é denominado vértice de conflito, se o mesmo possui dois vértices antecedentes que definem assinalamentos distintos para a mesma variável.

Um procedimento simples para gerar uma cláusula de conflito através de um grafo de implicações como esse é mostrado nas seguintes definições:

Definição 4 *Seja $G = (V, A)$ um grafo de implicações para análise de conflitos. Um corte ψ é uma operação que transforma o grafo G em um grafo bipartido G', que é composto por uma partição com os vértices de conflito e outra partição com os vértices de decisão, denominadas P_{causa} e $P_{conflito}$, respectivamente.*

[4]Existem diversos tipos de grafos de implicações de acordo com o contexto utilizado. No presente trabalho, o grafo é utilizado em dois contextos: o de satisfabilidade e o de derivação de implicações. O primeiro será referido como Grafo de Implicações para análise de conflitos e o segundo referido por apenas Grafo de Implicações, já que o mesmo é utilizado na maior parte desse trabalho.

Definição 5 *Seja $G = (V, A)$ um grafo de implicações para análise de conflitos. Uma cláusula de conflito ω_c é criada a partir da negação da conjunção das fases*[5] *dos assinalamentos de um conjunto de vértices U formado por todos os vértices u em que existe uma aresta (u, v) tal que $u \in P_{causa}$ e $v \in P_{conflito}$.*

A Figura 2.1 mostra um exemplo de um grafo de implicações para análise de conflitos. Como afirmado anteriormente, cada vértice corresponde a um assinalamento. Os vértices pretos representam assinalamentos gerados pela função de decisão. Os vértices brancos representam assinalamentos gerados por implicações através da função de dedução. O vértice cinza é o vértice especial que indica que houve um conflito.

Cada vértice possui um assinalamento do tipo $x_n = v$ @ m, em que x_n é uma variável da fórmula φ, v é o valor do seu assinalamento ($v \in \{0, 1\}$) e m é o nível da árvore na qual foi feita a decisão. Note que todas as implicações do grafo foram disparadas pelo assinalamento que está em negrito $x_5 = 0$ @ 9 e todas as implicações estão no mesmo nível.

No grafo de implicações da Figura 2.1 existem três cortes distintos ψ_1, ψ_2 e ψ_3 que criam as partições P_{causa} e $P_{conflito}$ assim como apresentado na Definição 4. Observe que, de acordo com o corte, alguns vértices brancos ora pertencem a partição P_{causa} e ora pertencem a partição $P_{conflito}$, mas sempre respeitando a Definição 4.

De acordo com a Definição 5, o conjunto de vértices U é formado por todos os vértices u em que existe uma aresta (u, v) tal que $u \in P_{causa}$ e $v \in P_{conflito}$, ou seja, é o conjunto dos vértices que possuem arestas saindo de P_{causa} e entrando em $P_{conflito}$. Então, na Figura 2.1, escolhendo-se o corte ψ_1, o conjunto U é dado por $U = \{x_2, x_5, x_7, x_1, x_4\}$ que representa o conjunto de assinalamentos A dado por $A = \{x_2 = 1, x_5 = 0, x_7 = 0, x_1 = 1, x_4 = 0\}$. Uma cláusula de conflito ω_{c1} criada a partir da negação da conjunção das fases de A seria:

$$\omega_{c1} = (\overline{x_2 \wedge \overline{x}_5 \wedge \overline{x}_7 \wedge x_1 \wedge \overline{x}_4})$$

O que em CNF equivale a:

$$\omega_{c1} = (\overline{x}_2 \vee x_5 \vee x_7 \vee \overline{x}_1 \vee x_4)$$

Ao adicionar a cláusula de conflito ω_c à fórmula original φ, evita-se que implicações tais quais as da Figura 2.1 sejam geradas novamente, evitando-se por conseqüência o conflito. Note que os cortes ψ_2 e ψ_3 gerariam as cláusulas de conflito ω_{c2} e ω_{c3} respectivamente: $\omega_{c2} = (\overline{x}_{11} \vee x_{14} \vee x_4)$ e $\omega_{c3} = (\overline{x}_{15} \vee x_4)$. Em uma análise mais superficial, a cláusula ω_{c3} pode ser considerada mais forte dos que as anteriores por

[5] Os termos fase positiva e fase negativa de uma variável estão para a área de Satisfabilidade assim como os termos variável e variável complementada estão para a área de circuitos.

possuir menos literais, já que a probabilidade de acontecer um par de assinalamentos é maior do que acontecer uma tupla com 5 assinalamentos. No entanto, isso pode não valer para qualquer instância[6].

Uma das principais conseqüências de se fazer um Aprendizado Direcionado por Conflito é que é possível fazer um retrocesso não cronológico na árvore de decisão. Por exemplo, a Figura 2.1 mostra o conflito gerado a partir do assinalamento $x_5 = 0$ @ 9. Utilizando-se um resolvedor de SAT com Retrocesso Cronológico, o próximo passo para solucionar o problema é complementar o assinalamento da decisão feita no último nível, ou seja, fazer $x_5 = 1$ @ 9; o que pode resultar em duas situações distintas.

A primeira delas é não acontecer nenhum conflito devido ao novo assinalamento. Daí, a busca prossegue em um nível mais profundo na árvore de decisão. No entanto, a segunda situação é a ocorrência de um conflito devido ao assinalamento. Se isto ocorrer, significa que a variável x_5 não é a responsável pelo conflito, já que o mesmo acontece independentemente do valor de assinalamento 0 ou 1 da variável x_5. Nesse caso, a partir de uma análise da situação, Silva e Sakallah [SS96] adaptaram, para a área de circuitos, uma solução interessante denominada Retrocesso Não Cronológico.

Suponha que a cláusula de conflito ω_{c1} tenha sido adicionada à fórmula original φ devido ao conflito durante o primeiro assinalamento ($x_5 = 0$). Como a variável x_5 não é a responsável pelo conflito, pois $x_5 = 1$ também resulta em conflito, algum dos assinalamentos que permaneceu constante no grafo de implicações ($x_2 = 1, x_7 = 0, x_1 = 1, x_4 = 0$) tem que ser o responsável.

O assinalamento da variável de decisão x_5 foi feito no nível $d = 9$ (veja a Figura 2.1). No grafo de implicações apresentado, a última variável decidida antes de x_5 é x_2 que teve o seu assinalamento no nível $d = 5$. Isso significa que, se um retrocesso for feito para o nível $d = 8$, o mesmo grafo de implicações será gerado e o mesmo conflito aparece. O mesmo acontece para os níveis $d = 7$ e $d = 6$. Somente em $d = 5$, o assinalamento de x_2 pode ser complementado e esse grafo de implicações deixa de existir. Dessa forma, a busca por uma solução nos níveis $d = 8$, $d = 7$ e $d = 6$ é inútil. Por isso, em detrimento de se fazer um retrocesso cronológico para o nível anterior, pode-se ganhar bastante tempo retrocedendo diretamente para $d = 5$. Este mecanismo possibilitou, pela primeira vez, a resolução de problemas com dezenas de milhares de variáveis em questão de horas[7].

[6]Existem diversos procedimentos para análise de conflito de acordo com o tipo de problema trabalhado. Para análises mais aprofundadas, consulte [SS96], [SS99] e [MMZ+01].

[7]Para mais informações sobre Retrocesso Não Cronológico e uma prova de que a técnica faz uma busca completa no espaço de soluções consulte [Sil95].

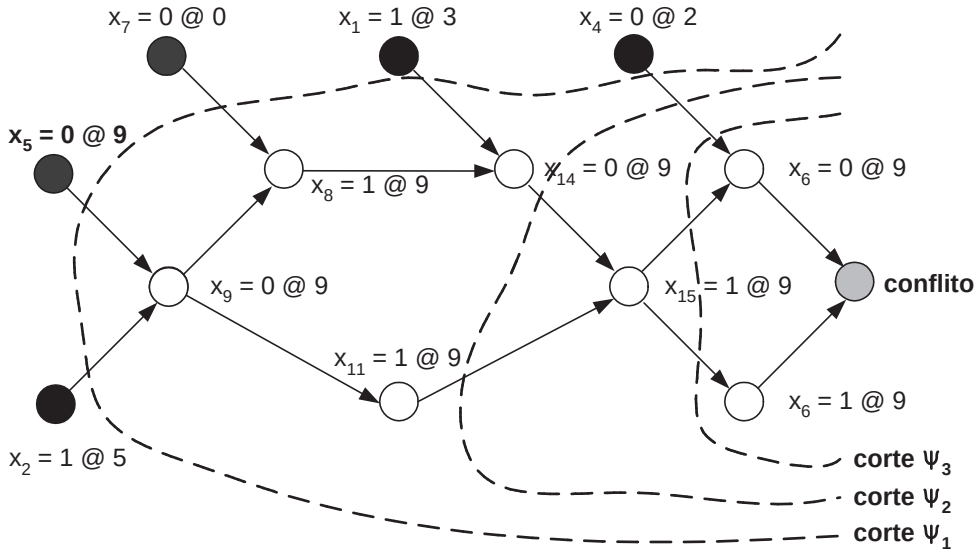

Figura 2.1: Exemplo de um grafo de implicações para análise de conflitos

Heurística de Decisão

As heurísticas de decisão em resolvedores de SAT são responsáveis pela ordem de escolha dos assinalamentos. No exemplo da Seção 2.2.3, a ordem de escolha dos assinalamentos foi a lexicográfica. Como a fórmula φ_1 é muito pequena, uma heurística baseada em outra ordem que não fosse a lexicográfica teria um desempenho similar. No entanto, em instâncias maiores, sejam derivadas de circuitos digitais ou de funções aleatórias, o impacto é muito grande, por isso, diversas heurísticas de decisão já foram propostas.

Como a operação de BCP é bastante eficiente, heurísticas de decisão que fazem com que essa operação seja ativada mais vezes podem trazer bons resultados. Assim funciona uma heurística que é conhecida como Máxima Ocorrência de Literais em Cláusulas de Tamanho Mínimo (MOM, do Inglês, *Maximum Occurrences on clauses of Minimum sizes*). Se uma cláusula de tamanho mínimo possuir uma de suas variáveis assinaladas, maior a probabilidade de que ela se torne uma cláusula com apenas um literal na qual o BCP pode ser aplicado. Uma pequena modificação na heurística MOM foi proposta por Jeroslow e Wang [JW90]. Esses autores sugeriram utilizar uma função com pesos em detrimento de apenas contar a ocorrência do literal nas cláusulas de tamanho mínimo. Tais abordagens funcionam bem para instâncias de SAT de funções aleatórias que não são o foco do presente trabalho.

Dentre as diversas propostas recentes[8], destaca-se o trabalho de Moskewicz et al. [MMZ+01] no resolvedor zChaff que obteve bons resultados principalmente em instâncias derivadas de circuitos. A heurística utilizada no zChaff é denominada Soma Decrescente Independente do Estado da Variável (VSIDS, do Inglês, *Variable State*

[8]Para uma boa revisão bibliográfica sobre o tema, consulte o trabalho de Silva [Sil99].

Independent Decaying Sum).

Explicando sucintamente, a heurística consiste em medir o nível de atividade de cada variável em suas fases, positiva e negativa, através de contadores. No início da busca, todos os contadores começam com valor 0. À medida que cláusulas de conflito são encontradas, as variáveis que participam do conflito têm os seus contadores incrementados e, a cada nível de decisão, todas as variáveis têm os contadores divididos por uma constante. A decisão de assinalamento recai sobre as variáveis que possuem contadores com valores maiores.

Um resolvedor de SAT mais recente denominado MiniSAT [ES03] já superou[9] os resultados obtidos com o resolvedor zChaff, utilizando a mesma heurística de decisão com uma pequena alteração. Diferentes constantes são utilizadas para incrementar e dividir os valores dos contadores; além disso, existem contadores apenas para os literais e não para cada uma de suas fases, positiva ou negativa.

2.3 Aprendizado em Circuitos Combinacionais

2.3.1 Introdução

Uma das primeiras técnicas bem sucedidas no Aprendizado em Circuitos Combinacionais foi proposta por Schulz et al. [STS88] e posteriormente aperfeiçoada por Kunz e Pradhan [KP93]. É interessante notar que o contexto inicial de aplicação do Aprendizado foi na área de ATPG, no entanto, o mesmo pode ser utilizado em outras áreas. Nesse trabalho, o Aprendizado em Circuitos Combinacionais é aplicado no problema de Verificação de Equivalência.

De forma mais ampla, o Aprendizado em Circuitos Combinacionais pode ser definido como um conjunto de técnicas que possibilita extrair informações de um circuito de modo a tornar uma determinada técnica mais eficiente. Essas informações são denominadas implicações e estão definidas na próxima seção.

2.3.2 Definições Básicas

Existem quatro tipos de implicação que podem ser derivadas através de Aprendizado em Circuitos: as implicações simples, diretas, indiretas e estendidas. Para entender como as implicações são derivadas é necessário estabelecer algumas definições.

[9] Apesar de já ter superado os resultados do zChaff em instâncias de benchmarks de verificação formal, é extremamente complexo avaliar se isso aconteceria se apenas a heurística de decisão fosse alterada. Sempre que um novo resolvedor de SAT torna-se público, o mesmo vem com estruturas de dados distintas, diferentes implementações para as principais heurísticas e diferentes heurísticas. Isso torna difícil avaliar o que realmente elevou o desempenho do resolvedor novo em relação aos anteriores.

Definição 6 *Seja G uma porta lógica que possui no mínimo uma entrada ou a sua saída especificada: a porta lógica G é denominada* ***não-justificada*** *se é possível encontrar algum assinalamento para essa porta que conduza a um conflito. Caso contrário, a porta é denominada* ***justificada****.*

Definição 7 *Um conjunto de assinalamentos $J = \{f_1 = V_1, f_2 = V_2, \ldots, f_n = V_n\}$, em que $f_1, f_2, \ldots, f_n$ são entradas ou a saída da porta lógica G, é denominado* ***justificação*** *de G caso a combinação de assinalamentos em J torne G justificada.*

Definição 8 *Seja G_C um conjunto de m justificações $J_1, J_2, \ldots, J_m$ para uma porta G não-justificada. Se existe ao menos uma justificação $J_i \in G_C$ para $i = 1, 2, \ldots, m$ para qualquer justificação possível J^* de G, tal que $J_i \subseteq J^*$, então G_C é denominado* ***conjunto completo de justificações****.*

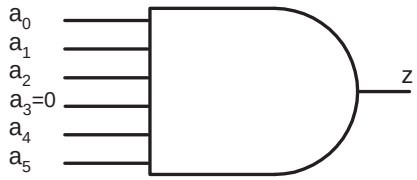

Figura 2.2: Exemplo de uma porta não-justificada

O exemplo seguinte ilustra melhor as Definições 6, 7 e 8. Suponha que o circuito da Figura 2.2 seja dado. A figura mostra uma porta lógica AND não-justificada, segundo a Definição 6. Já que o assinalamento $z = 1$ conduz essa porta a um conflito, pois o assinalamento $a_3 = 0$ necessariamente leva $z = 0$. Dessa forma, segundo a Definição 7, uma justificação para essa porta seria $J = \{a_1 = 0, a_2 = 0, a_4 = 0, a_5 = 0, z = 0\}$. Outras justificações podem ser obtidas fazendo todas as combinações de assinalamentos possíveis entre a_1, a_2, a_4 e a_5, e adicionando sempre $z = 0$, pois, o assinalamento de 0 em z já é suficiente para fazer a justificação da porta. Por esse motivo, o conjunto completo de justificações é dado por $G_C = \{J_1\}$ em que $J_1 = \{z = 0\}$. Note que a justificação J_1 inclui qualquer combinação de assinalamentos para as entradas a_1, a_2, a_4 e a_5, pois as mesmas não estão na justificação.

Por meio das Definições 6, 7 e 8, é possível diferenciar entre implicações simples, diretas, indiretas e estendidas como mostrado a seguir.

Definição 9 *Uma* ***implicação simples*** *é uma implicação da forma $a \rightarrow b$ em que a é um assinalamento escolhido e b é um assinalamento obtido por meio da propagação do sinal a em apenas um nível de portas lógicas (uma porta lógica).*

Definição 10 *Uma* ***implicação direta*** *é uma implicação da forma $a \rightarrow b$ em que a é um assinalamento escolhido e b é um assinalamento obtido pela propagação do sinal a em mais de um nível de portas lógicas (mais de uma porta lógica).*

Implicações indiretas são baseadas em justificação de portas lógicas [KP94]. Portanto:

Definição 11 *Uma **implicação indireta** é uma implicação da forma $a \rightarrow b$ em que a é um assinalamento na saída de uma porta lógica não justificada com sinais de entrada não especificados; e b é qualquer assinalamento resultante da interseção de todos os assinalamentos obtidos através de implicações simples e diretas para cada justificação possível.*

Finalmente, assim como definido por Zhao et al. [ZNP97],

Definição 12 *Uma **implicação estendida** é o conjunto de implicações indiretas que pode ser derivado a partir da lista de implicações de uma implicação direta.*

A Figura 2.3 ilustra como implicações simples e diretas são derivadas em um circuito. A figura mostra um trecho de circuito qualquer que possui entradas de i_0 a i_3, saída z e portas lógicas de a a i. Seja $z_h = 1$ um assinalamento na saída da porta lógica h[10], uma implicação simples gera o assinalamento $z_e = 1$, já que o sinal passou apenas por um nível de portas lógicas, isto é, uma porta lógica. Uma implicação direta gera o assinalamento $z_b = 0$. Os únicos assinalamentos que podem ser deduzidos a partir de $z_h = 1$ são $z_e = 1$ e $i_3 = 1$; e $z_b = 0$, gerando, assim, as seguintes implicações $z_h \rightarrow z_e$, $z_h \rightarrow i_3$ e $z_h \rightarrow \bar{z_b}$.

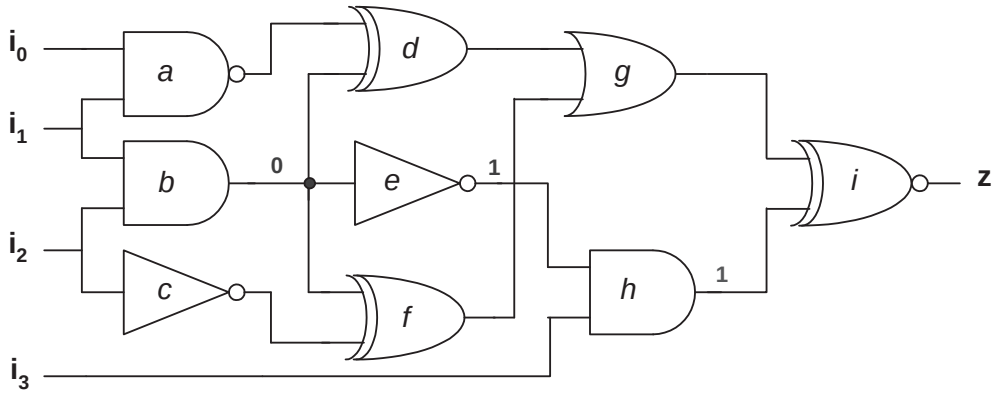

Figura 2.3: Exemplo de implicações simples e diretas

A Figura 2.4 mostra como implicações indiretas e estendidas podem ser derivadas em um circuito. A partir das implicações diretas obtidas anteriormente, é possível obter novas implicações a partir da justificação das portas. Observe a porta lógica b. O seu conjunto completo de justificações é dado por $G_C = \{J_1, J_2\}$, em que $J_1 = \{a_b = 0\}$ e $J_2 = \{b_b = 0\}$. A Figura 2.4 ilustra os dois casos (desenho inferior e desenho superior).

[10] A saída de uma porta lógica x será sempre denominada z_x e as suas entradas denominadas a_x e b_x. Esses rótulos não foram inseridos na figura por questão de legibilidade.

Para a justificação J_1 (desenho superior), apenas simulando a tabela-verdade de cada porta lógica, pode-se obter a seguinte cadeia de assinalamentos: $z_a = 1$, $z_d = 1$, $z_g = 1$ e $z_i = 1$ ($z = 1$). Para a justificação J_2 (desenho inferior), seguindo o mesmo procedimento, pode-se obter a seguinte cadeia de assinalamentos: $z_c = 1$, $z_f = 1$, $z_g = 1$ e $z_i = 1$ ($z = 1$).

A partir do conjunto completo de justificações G_C, pode-se derivar diversas implicações indiretas. Segundo a Definição 11, as implicações indiretas correspondem à interseção entre todos os conjuntos de assinalamentos obtidos a partir de cada justificação pertencente a G_C, isto é, $z_g = 1$ e $z_i = 1$ ($z = 1$). Os assinalamentos $z_a = 1$, $z_c = 1$, $z_d = 1$ e $z_f = 1$ só acontecem em uma das justificações. Assim, podem-se gerar implicações indiretas do tipo $\bar{z_b} \rightarrow z$. De acordo com a Definição 12, $z_e \rightarrow z$ e $z_h \rightarrow z$ são implicações estendidas, uma vez que as mesmas são implicações indiretas obtidas através da lista de implicações diretas da justificação de uma porta.

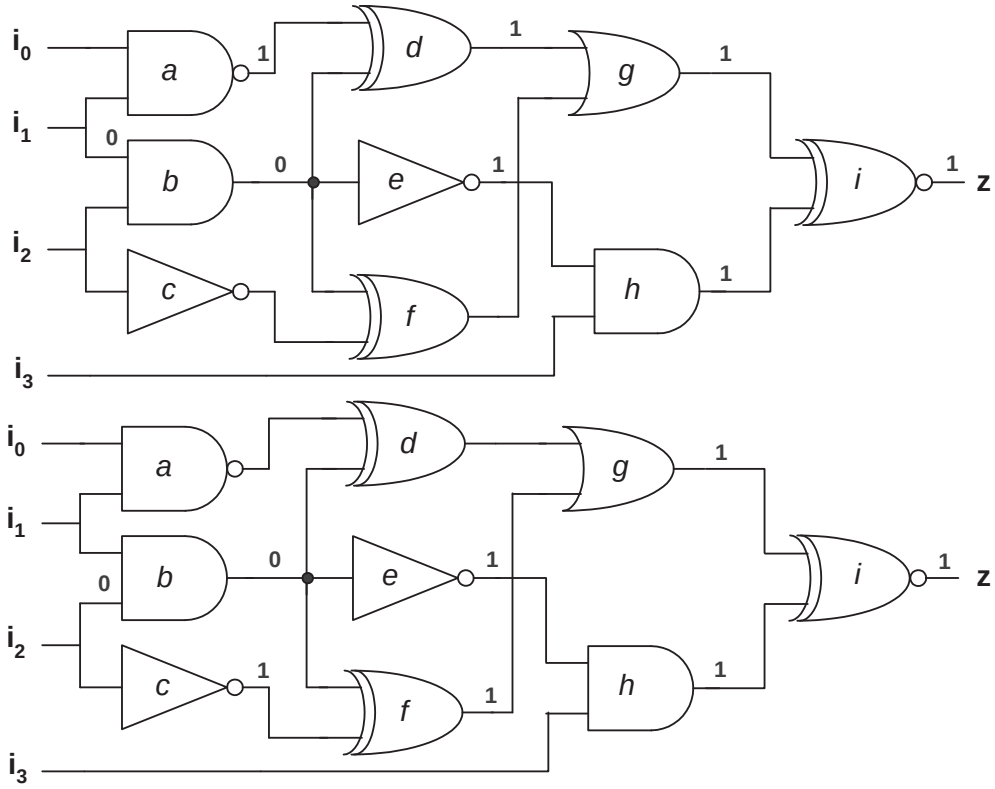

Figura 2.4: Exemplo de implicações indiretas e estendidas

2.4 Grafo de Implicações

Essa seção apresenta uma estrutura de dados muito importante para a criação de ferramentas de derivação de implicações: o grafo de implicações.

Um grafo de implicações é um grafo direcionado que representa a relação de implicações entre variáveis, tal grafo pode ser completo ou parcial de acordo com os tipos

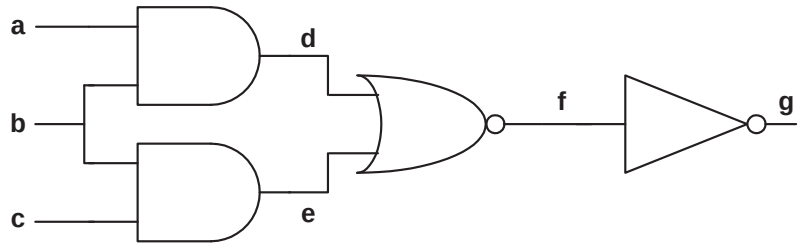

Figura 2.5: Circuito que origina o grafo de implicações da Figura 2.6

de implicações que o mesmo representa.

Um grafo de implicações completo é um grafo direcionado $G(V, A)$ em que V é um conjunto de vértices e A é um conjunto de arestas. Cada vértice é rotulado como um vértice de sinal ou como um vértice AND. Vértices de sinal são rotulados com a fase positiva ou a fase negativa de uma variável qualquer. Já os vértices AND não são rotulados, pois representam apenas a conjunção de duas variáveis. Uma aresta entre dois vértices representa uma implicação da variável existente no vértice fonte para a variável do vértice destino.

Um vértice de sinal é denominado *ativado* quando o mesmo possui algum de seus vértices predecessores como ativado ou quando o mesmo foi escolhido como ativado. Dessa forma, a partir da escolha de um vértice ativado qualquer, uma sucessão de vértices ativados pode ser encontrado. Em contrapartida, um vértice AND é denominado ativado somente quando o mesmo possui todos os seus vértices predecessores ativados ou ainda quando o mesmo foi escolhido como ativado.

Um grafo de implicações parcial segue as mesmas definições do grafo de implicações completo, a única diferença é que o grafo de implicações parcial não possui vértices AND.

Para se construir um grafo de implicações a partir de um circuito basta seguir os seguintes passos. Primeiramente, converte-se cada porta para a sua fórmula em CNF. Em seguida, descreve-se cada cláusula encontrada por meio de implicações. Finalmente, cria-se um vértice para cada literal da implicação e uma aresta direcionada no sentido da implicação.

Para ilustrar, a Figura 2.6 mostra um grafo de implicações completo para o circuito da Figura 2.5. O assinalamento 0 e 1 em cada nó n do circuito é representado através de $-n$ e n no grafo de implicações respectivamente. Os vértices circulares representam os vértices de sinal e os vértices triangulares representam os vértices AND. As linhas contínuas conectam vértices de sinal e as linhas tracejadas conectam vértices AND a vértices de sinal e vice-versa.

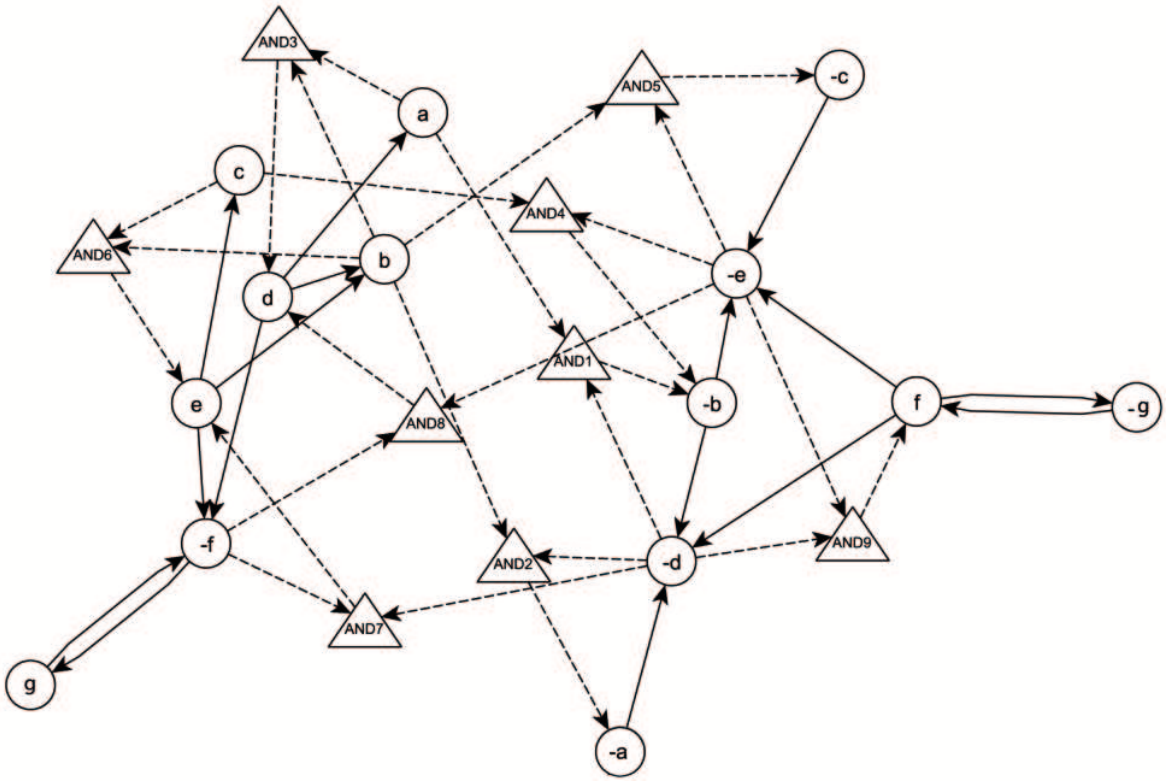

Figura 2.6: Grafo de implicações referente ao circuito da Figura 2.5

Capítulo 3

Verificação de Equivalência de Circuitos Combinacionais

3.1 Introdução

Este capítulo fornece uma revisão bibliográfica das diferentes metodologias e técnicas utilizadas na Verificação de Equivalência de Circuitos Combinacionais (CEC, do Inglês *Combinational Equivalence Checking*). O problema de CEC consiste em provar que dois circuitos apresentam os mesmos valores de saída quando submetidos às mesmas entradas. Na maior parte dos casos, a descrição dos circuitos é fornecida em nível de portas lógicas. Entretanto, também pode ser fornecida em Nível de Transferência de Registros (RTL, do Inglês *Register Transfer Level*) para ambos circuitos, ou em nível de portas lógicas para um deles e RTL para o outro. A CEC, apesar de estar na classe de problemas co-NP-difícil, já se tornou viável mesmo para instâncias com dezenas de milhares de portas lógicas. Isso acontece porque os problemas de CEC originados na indústria não são baseados em funções arbitrárias, mas sim baseados em funções específicas derivadas de circuitos integrados.

Ao longo de mais de 20 anos, diversas metodologias foram propostas. As principais metodologias são baseadas nas seguintes técnicas ou formalismos:

- Geração Automática de Padrões de Testes (ATPG, do Inglês, *Automatic Test Pattern Generation*);
- Diagrama de Decisão Binária (BDD, do Inglês, *Binary Decision Diagram*);
- Diagrama de Momento Binário (BMD, do Inglês, *Binary Moment Diagram*);
- Aprendizado Recursivo (RL, do Inglês, *Recursive Learning*);
- Resolvedores de Satisfabilidade (SAT).

Os principais trabalhos de cada um dos formalismos são apresentados nas seções seguintes. Todavia, devido ao grande número de variações existentes na maneira de se lidar com o problema de CEC, é necessário estabelecer algumas classificações.

3.2 Classificações para CEC

O problema de CEC pode ser classificado de acordo com diversas propriedades: motores de resolução (também chamado de técnicas), tipos de particionamento, características dos circuitos verificados, dentre outros. O estabelecimento da classificação é necessário para se contextualizar o escopo do presente trabalho.

3.2.1 Motores de Resolução para CEC

Os diferentes trabalhos existentes sobre CEC podem ser divididos de acordo com o motor de resolução utilizado. Os motores de resolução mais populares são:

Motor ATPG: baseia-se na construção do *miter* e da prova de que os circuitos são equivalentes através de uma falha do tipo *stuck-at-0*. Para se tornar mais eficiente, normalmente, subfunções do circuito são provadas equivalentes, reduzindo o tamanho do problema. Trabalha no nível de portas lógicas (ver detalhes na Seção 3.4).

Motor BDD: é utilizado para representar circuitos no nível de portas lógicas a serem provados equivalentes por meio da propriedade da canonicidade do BDD, quando esse possui suas variáveis ordenadas e é reduzido (ver detalhes na Seção 3.5).

Motor BMD: é utilizado para representar circuitos (geralmente aritméticos) em um processo de Verificação de Equivalência entre uma descrição em um nível mais alto de abstração e um nível de portas lógicas. Necessita-se que uma especificação em alto nível seja fornecida, o que nem sempre acontece (ver detalhes na Seção 3.5).

Motor RL: executa uma busca sobre a estrutura topológica do circuito com o intuito de derivar implicações entre sinais do mesmo circuito. Esse motor geralmente é utilizado em problemas em nível de portas lógicas (ver detalhes na Seção 3.6).

Motor SAT: como motor de resolução único, é bastante recente e só conseguiu maior destaque devido aos grandes avanços dos resolvedores de SAT na última década. Baseia-se na prova de propriedade na saída do *miter* para garantir a equivalência entre os circuitos. Geralmente, trabalha com a descrição dos circuitos em Forma Normal Conjuntiva (CNF, do Inglês, *Conjunctive Normal Form*) e a maior parte

das metodologias que utilizam esse motor não usam informações topológicas do circuito (que é perdida na transformação do mesmo para CNF). Além disso, o motor SAT pode aprender relações mais gerais sobre os circuitos e trabalha geralmente na CEC em nível de portas lógicas (ver detalhes na Seção 3.7).

Motor Híbrido: existem diversas categorias que trabalham com motores híbridos, dentre estas as principais são:

BDD e SAT: existem diversas possibilidades de se integrar os motores BDD e SAT. Uma das mais antigas consiste em utilizar SAT para eliminar falsos negativos. Existem também alternativas que utilizam a ordenação de variáveis no BDD para definir a ordem dos assinalamentos nas variáveis do mecanismo de decisão do resolvedor de SAT.

SAT e RL: uma das principais abordagens desse motor híbrido consiste em executar um RL durante o motor de decisão com o intuito de descobrir relações que dificilmente seriam encontradas por meio de BCP (do Inglês, *Boolean Constraint Propagation*).

BDD, SAT, RL e Simulação: é uma abordagem que integra os principais motores de resolução. A mais antiga dessas abordagens tenta encontrar informações específicas dos circuitos de modo a determinar a sua classe e submetê-lo ao motor mais adequado. Se o circuito possuir muitas similaridades, utilizam-se os motores BDD ou RL. Se o circuito não apresentar similaridades o suficiente, utiliza-se o motor SAT. Caso o circuito seja pequeno, simulação pode ser a mais recomendada. Outras abordagens consistem em fazer com que a ferramenta troque de motor durante a resolução do problema e transfira informações obtidas com o motor atual para o motor seguinte.

Motor Provadores: utiliza técnicas de prova de teoremas para garantir a exatidão do circuito. Não é tão popular quanto os motores descritos anteriormente e trabalha no nível de portas lógicas.

3.2.2 Características Internas dos Circuitos sob CEC

O par de circuitos submetidos à CEC pode ser classificado de acordo com o grau de similaridade entre eles. As duas principais classes são:

Circuitos Similares: os dois circuitos possuem um alto grau de similaridades estruturais entre si. Circuitos com um alto grau de similaridades estruturais são gerados, principalmente, quando há uma intervenção manual nos circuitos ou quando ferramentas de síntese mais simples são utilizadas. A maior parte das

metodologias para a CEC leva em consideração a similaridade entre os dois circuitos, principalmente as baseadas em BDD e RL.

Circuitos Dissimilares: os dois circuitos possuem pouquíssimas ou nenhuma similaridade interna. Podem ser gerados por ferramentas de síntese altamente eficientes, mecanismos de otimização ou ferramentas que utilizem blocos de circuitos aritméticos que podem ser instanciados. Acredita-se que o processo de síntese atual poderia ser muito melhor, caso a restrição de similaridade entre o circuito original e o circuito sintetizado fosse ignorada [GN03].

Os circuitos também podem ser classificados de acordo com a estrutura interna em outros três aspectos:

Circuitos com Dependência Estrutural : possuem os bits das saídas primárias altamente dependentes do circuito lógico utilizado para gerar os bits das saídas primárias anteriores. São bastante comuns em circuitos aritméticos.

Circuitos com Interseção Estrutural : possuem bits das saídas primárias que dependem de parte do circuito lógico utilizado para gerar os bits de outras saídas primárias.

Circuitos sem Interseção Estrutural : possuem saídas com lógicas independentes entre elas. Geralmente são circuitos combinacionais simples como: decodificadores, codificadores, multiplexadores, demultiplexadores e comparadores.

3.2.3 Particionamento dos Circuitos para CEC

Os circuitos a serem verificados podem ser tratados de três formas distintas em relação ao particionamento das saídas:

Circuitos Não Particionados. O circuito é verificado como um todo, isto é, com todas as entradas e saídas primárias de uma só vez.

Circuitos Particionados pelas Saídas Primárias. O circuito é particionado de acordo com o Fan-In Transitivo (TFI, do Inglês, *Transitive Fan-In*) das saídas primárias. Cada uma das saídas primárias é verificada de forma independente em relação às outras. Em grande parte dos casos, o particionamento das saídas acontece das saídas primárias menos significativas para as mais significativas.

Circuitos Particionados pelas Saídas Primárias Agrupadas. O circuito é particionado de acordo com uma análise na sua estrutura interna que define quais saídas devem ser agrupadas em nível de palavras para aumentar o desempenho do processo de verificação.

Subparticionamentos também podem ocorrer dentro dos particionamentos descritos anteriormente. O principal deles é denominado corte e é definido como:

Corte: é aplicado dentro de partições, principalmente em partições feitas pelas saídas primárias, com o intuito de reduzir ainda mais o tamanho do problema de CEC.

3.3 Heurísticas de Particionamento e Cortes

A principal heurística de particionamento é a baseada no TFI. O TFI é o Fan-In aplicado recursivamente a partir de uma porta lógica até a entrada primária do circuito. No caso de CEC, o TFI é freqüentemente aplicado à saída do *miter*. A Figura 3.1 mostra um circuito com entradas de a a h e saídas z_0 a z_2. O conjunto de portas hachuradas corresponde às portas do TFI de z_1.

Uma outra forma de se obter o TFI é representar o circuito por um grafo direcionado $G = (V, A)$, em que V é um conjunto de vértices e A é um conjunto de arestas. Cada vértice representa uma porta lógica e cada aresta é direcionada no sentido da propagação do sinal. O TFI pode ser obtido invertendo todas as arestas e obtendo os componentes conectados a partir da aresta da saída primária escolhida.

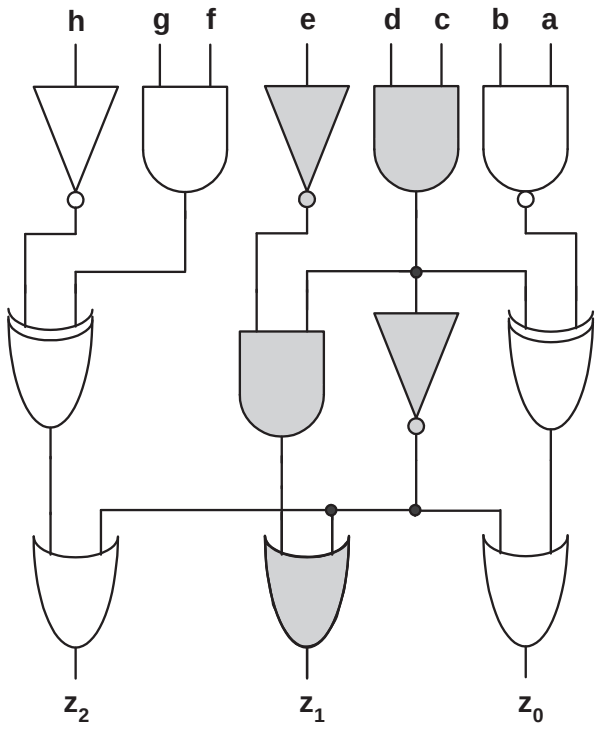

Figura 3.1: Exemplo de Fan-In Transitivo

Em casos de circuitos Estruturalmente Dependentes, o circuito gerado pelo particionamento utilizando o TFI pode se tornar algo interessante. Nesses casos, o circuito lógico que define a saída $i + 1$ depende de todo o circuito lógico utilizado para se produzir a saída i e assim sucessivamente. Portanto, fica claro que a partição gerada a

partir do TFI do bit de saída mais significativo inclui todo o circuito dessa saída até a saída do bit menos significativo. Para esses circuitos, o particionamento pelo TFI não é suficiente e, por isso, foi necessário desenvolver técnicas que faziam subparticionamentos as quais foram denominadas *cortes*. O exemplo a seguir mostra como cortes podem ser feitos em dois circuitos sob CEC.

Sejam duas funções f e f' que foram implementadas em dois circuitos C e C' respectivamente. Ambos os circuitos possuem apenas uma saída primária denominada z e z' e pontos internos denominados x e x'. Suponha que um corte (corte 1) tenha sido feito em cada circuito justamente sobre os pontos x e x', como mostrado na Figura 3.2. Nessa situação, esses pontos podem ser encarados como saídas primárias de um circuito menor definidos por g e g' que são subfunções de f e f' respectivamente. Assim, pode-se resolver a instância do problema de CEC para as subfunções g e g' que é uma instância menor do que para as funções f e f'.

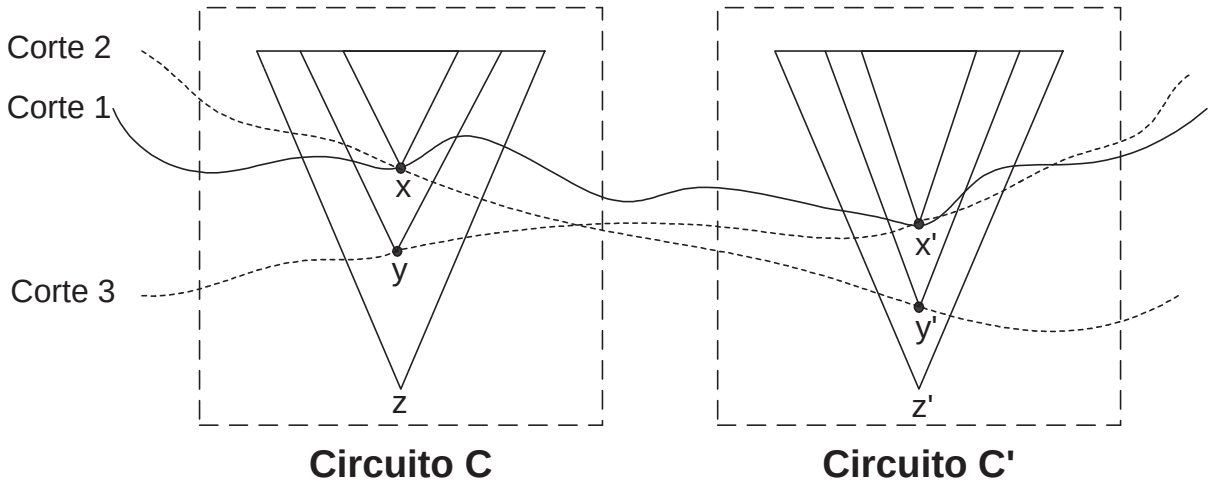

Figura 3.2: Exemplo de cortes em circuitos sob CEC

Se o problema de CEC de g e g' retornar que essas funções são equivalentes, o trecho do circuito de g e g' poderá ser retirado do problema de verificação de f e f' apenas fazendo com que x e x' sejam novas entradas primárias. Note que, em certos casos, isso pode reduzir drasticamente o tamanho do problema, se for aplicado de forma sucessiva.

Entretanto, se o problema de CEC de g e g' retornar que as subfunções não são equivalentes, não implica necessariamente que f e f' não são equivalentes. Suponha que os circuitos acima tenham também pontos y e y' que são equivalentes e distintos de x e x'. Suponha também que outro corte tenha gerado subfunções com saídas primárias y e x' ou y' e x, como mostrado na Figura 3.2, cortes 1 e 2. Certamente, esses pontos não são equivalentes, porque o corte não foi aplicado no ponto exato para separar y e y' ou x e x'. Esse efeito é denominado *falso negativo* pois, a princípio, a ferramenta de verificação poderia indicar que os circuitos C e C' não são equivalentes. Quando isso ocorre, quase na totalidade dos casos, é necessário trabalhar com a partição inteira com

o intuito de provar que os circuitos são equivalentes; o que limita bastante a eficiência dos motores de resolução que trabalham com técnicas de corte.

3.4 CEC Utilizando ATPG

Geração Automática de Padrões de Teste para circuitos combinacionais é uma das técnicas mais antigas para se garantir a exatidão de circuitos integrados. O método consiste em gerar um padrão de teste que seja capaz de demonstrar falhas no circuito sob verificação, quando esse é comparado com o circuito de referência. Desta forma, quando o padrão de teste é aplicado às entradas primárias de um circuito qualquer, os valores das saídas primárias do circuito correto devem ser diferentes do circuito com erro.

No entanto, modelar qualquer tipo de falha existente em qualquer tipo de circuito combinacional é uma tarefa árdua e intratável. Por isso, para os tipos de falhas mais freqüentes, um tratamento mais específico é feito o que é denominado modelo de falhas. Um dos modelos de falhas muito utilizado é o modelo de falhas do tipo *stuck-at*. Uma falha do tipo *stuck-at* indica que um nó do circuito permaneceu no valor 0 para qualquer padrão de entrada (*stuck-at-0*), ou que o nó permaneceu no valor 1 (*stuck-at-1*).

Para garantir que um circuito não possua nenhuma falha do tipo *stuck-at-0*, é necessário verificar se determinado ponto no circuito é testável para a falha. Se for, significa que existe um padrão de teste que leva o ponto em inspeção da falha para o valor 1, indicando que o circuito está correto para a falha e a falha é observável. Apesar de representar uma enorme simplificação a respeito do conjunto de todas as falhas possíveis, esse modelo apresenta ótimos resultados na prática.

Para utilizar as técnicas de ATPG em CEC, é necessário construir um *miter* que pode ser definido da seguinte forma. Sejam dois circuitos C e C' que serão submetidos às mesmas entradas e que possuem saídas $z_0, z_1, z_2, \ldots z_n$ e $z'_0, z'_1, z'_2, \ldots z'_n$ respectivamente. Um *miter* é construído criando-se uma porta lógica XOR para cada par de saídas z_i e z'_i para todo $0 \leq i \leq n$ produzindo saídas $z_{m0}, z_{m1}, z_{m2}, \ldots, z_{mn}$. Cada uma das saídas ($z_{mi}$ para $0 \leq i \leq n$) é ligada a uma das entradas de uma porta lógica OR que, por sua vez, produz uma saída z que é denominada saída do *miter*. A Figura 3.3 mostra como um *miter* é construído para provar equivalência entre dois circuitos com sete saídas z_0 a z_6 e z'_0 a z'_6.

Para que a saída do *miter* alcance o valor lógico 1, é necessário que, pelo menos, uma das entradas da porta OR também tenha alcançado um valor lógico 1. Em conseqüência, no mínimo, uma porta lógica XOR teve em sua saída o valor lógico 1. Para que uma porta lógica XOR tenha o valor lógico 1 em sua saída, é necessário que essa possua entradas de valores distintos. Então, pelo menos um par de saídas dos circuitos C e C' alcançou valores distintos. Assim, sempre que o *miter* alcançar o valor lógico 1

em sua saída significa que os circuitos sob CEC não são equivalentes.

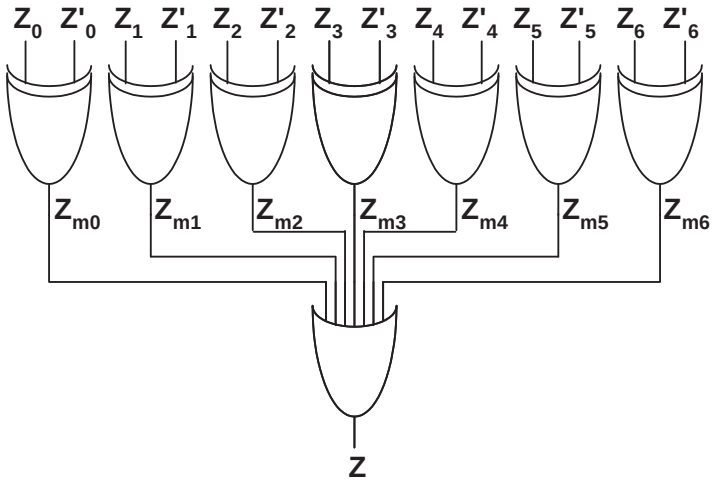

Figura 3.3: Exemplo de um circuito *miter* para dois circuitos com sete saídas primárias

No contexto de ATPG para problemas de CEC, o circuito *miter* é utilizado da seguinte forma. Para verificar a equivalência entre os dois circuitos, basta verificar se a saída do *miter* é testável para falhas *stuck-at*. Se for testável para uma falha *stuck-at-0*, então existe um padrão que faz com que a saída alcance um valor 1, conseqüentemente, os circuitos que compõem o *miter* não são equivalentes. Se não for testável para a mesma falha, os circuitos são equivalentes.

Esta metodologia obteve ótimos resultados e foi proposta por Brand [Bra93]. No entanto, para deixar o procedimento mais eficiente, o autor utiliza um conceito denominado substituição (*replaceability*) para aproveitar similaridades internas. Em detrimento de se criar um *miter* para o circuito inteiro, cria-se um *miter* para subfunções internas dos dois circuitos. Assim que ambas as funções são provadas equivalentes, a subfunção de um dos circuitos é substituída pela subfunção do outro, tornando ambos mais similares e mais fáceis de serem verificados. Essa técnica é similar a técnica de cortes apresentada na Seção 3.3. Entretanto, esse procedimento não é escalável e, por isso, à medida que o tamanho dos circuitos existentes foi aumentando drasticamente, a metodologia foi sendo abandonada.

3.5 CEC Utilizando BDD e BMD

O BDD e, em seguida, o BMD foram os principais impulsionadores do aumento de desempenho das ferramentas de CEC. Os primeiros trabalhos a receber um maior destaque utilizando o motor BDD foram o de Fujita et al. [FFK88] e Malik et al. [MWBSV88]. No entanto, até a publicação de Burch [Bur91], o caso mais patológico em relação à resolução de instâncias de problemas de CEC com o motor BDD ainda não havia sido tratado: a verificação de multiplicadores de números inteiros. No mesmo

ano da publicação de Burch [Bur91], Bryant [Bry91] provou que, para qualquer ordenação de variáveis em um BDD que representa um multiplicador de números inteiros, o tamanho do BDD cresce de forma exponencial em relação ao número de bits do multiplicador. Ainda assim, Burch [Bur91] alcançou bons resultados verificando multiplicadores de números inteiros por meio do motor BDD.

O autor conseguiu, pela primeira vez, verificar a equivalência entre um multiplicador de 16 x 16 bits com a sua cópia redundante através da sua técnica denominada *fanout splitting*. A técnica consiste em adicionar ao BDD variáveis que não existem no circuito original, mas que podem ser geradas a partir do mesmo. No seu artigo, o autor prova que o tamanho do BDD deixa de ter um crescimento exponencial para ter um crescimento de $O(n^3)$. Infelizmente, a técnica é muito específica, não funcionando para qualquer classe de circuito.

Procurando por uma técnica mais geral, van Eijk e Janssen [vEJ94] propuseram o que seria uma das técnicas mais utilizadas nos trabalhos vindouros: a utilização de similaridades entre os circuitos a serem provados equivalentes[1]. A idéia, apesar de simples, resultou em um ganho de desempenho bastante elevado. Devido à dificuldade de se representar um circuito completo por meio de apenas uma função utilizando BDDs, van Eijk e Janssen [vEJ94] propuseram que apenas subfunções do circuito deveriam ser construídas e provadas equivalentes. Isso seria feito utilizando-se a técnica de cortes apresentada na Seção 3.3. Para encontrar essas subfunções, bastaria submeter o circuito *miter* a uma simulação para a identificação de nós candidatos a serem as saídas das subfunções. Esse trabalho também foi um dos primeiros a adaptar de modo eficiente o mecanismo de (re)ordenação dinâmica de variáveis no BDD proposto por Rudell [Rud93].

Apesar de todo o aprimoramento na resolução de instâncias de problemas de CEC utilizando BDD, apenas Burch [Bur91] havia conseguido algum resultado razoável para a classe de circuitos aritméticos. Assim, Bryant e Chen [BC95] propuseram um mecanismo para a verificação de circuitos aritméticos que foi denominado BMD. A proposta surgiu da observação de que o BDD, por utilizar representações em nível de bits, necessitava de espaço exponencial e, para evitar que isso ocorresse, o BMD representaria as informações em nível de palavras, ou seja, uma representação em um nível mais alto de abstração.

Por meio de BMD, resultados antes considerados intratáveis foram obtidos, como a Verificação de Equivalência de um multiplicador de 256 x 256 bits em nível de portas lógicas a partir da sua especificação em nível de palavras em aproximadamente 18 minutos [BC95]. Entretanto, os resultados não podem ser comparados diretamente

[1]Isto no contexto de BDD, pois um ano antes da utilização de similaridades com técnicas de BDD, similaridades estruturais foram usadas para aumentar a eficiência de métodos baseados em RL [Kun93].

com as técnicas propostas anteriormente, já que as primeiras faziam a CEC entre dois circuitos em nível de portas lógicas e a última faz a CEC entre um circuito em alto nível e um circuito em nível de portas lógicas. Claramente, são dois problemas de CEC distintos. Além disso, o BMD necessita de descrições do circuito em nível de palavras que, quase sempre, não existem no processo de desenvolvimento do circuito integrado.

Alguns autores ainda continuam a obter excelentes resultados utilizando o BMD e implementando extensões a essa metodologia. Hamaguchi et al. [HMY95] melhoraram uma das deficiências nativas do BMD que consistia na ineficiência da construção do BMD para circuitos aritméticos, Keim et al. [KDB+97] provaram que, utilizando o BMD, é possível fazer CEC de multiplicadores *Wallace Tree* em tempo polinomial em relação ao número de bits do multiplicador, Chen e Chen [CC01] propuseram uma extensão ao BMD denominada PHDDs (do Inglês, *Multiplicative Power Hybrid Decision Diagrams*) que possibilitou a verificação de circuitos de multiplicadores dissimilares em menos de 1 minuto.

Apesar de todos os resultados excepcionais, a comunidade ainda continua melhorando a eficiência de CEC com o motor BDD. O principal argumento é que o BMD e o BDD trabalham em diferentes níveis de abstração e, por isso, não podem ser comparados diretamente. É interessante notar que grande parte das publicações, que utiliza o BMD, insiste em comparar os seus resultados com os obtidos com o BDD, enquanto publicações sobre BDD, geralmente, ignoram a existência do BMD nas seções de resultados.

Um trabalho que obteve ótimos resultados em CEC utilizando o motor BDD é o de Kuehlmann e Krohm [KK97]. Os autores propuseram uma estrutura de dados altamente eficiente (*circuit graph hashing*), eliminação controlada de falsos negativos e uma nova heurística de cortes para o circuito. Outro trabalho que obteve bons resultados mais recentemente é o de Xu [XYLG03] que propuseram uma nova heurística de cortes denominada *cortes independentes*.

As novas tendências na utilização do motor BDD em CEC consiste em integrá-lo às ferramentas que trabalham com múltiplos motores ou motores híbridos, como está descrito na Seção 3.8.

Considerando todas as técnicas descritas com o motor BDD, pode-se concluir que esse motor é bastante adequado quando se trabalha com circuitos não aritméticos, quando ambos os circuitos submetidos a CEC possuem um alto grau de similaridades ou quando os dois circuitos sob CEC são realmente equivalentes.

3.6 CEC Utilizando RL

A técnica de RL foi originalmente proposta por Kunz [Kun93] e Kunz e Pradhan [KP94] e foi criada a partir das técnicas de Aprendizado em Circuitos Combinacionais (Seção

2.3), ou seja, derivação de implicações entre sinais de um circuito. Essas implicações são utilizadas para provar a equivalência entre os dois circuitos que compõem o *miter* através de um procedimento descrito a seguir.

O primeiro passo do procedimento consiste na justificação (ver Seção 2.3) de todas as portas lógicas pertencentes ao TFI de uma saída escolhida. Durante a justificação de uma porta lógica, novas portas podem se tornar não justificadas sendo necessárias novas justificações e, assim, sucessivamente. Esses casos são tratados de forma recursiva. O procedimento utiliza o nível de recursão inicial igual a 1.

A justificação acontece através da atribuição do valor lógico 0 nas saídas das portas lógicas AND e NOR e através da atribuição do valor lógico 1 nas saídas das portas lógicas OR e NAND. Por meio das justificações, diversas implicações diretas e indiretas são derivadas e armazenadas.

Após derivar e armazenar as implicações para os dois circuitos que compõem *miter*, tenta-se justificar a saída do mesmo com o valor lógico 1, isto é, tenta-se encontrar um conjunto de assinalamentos que prove que a saída do *miter* pode alcançar o valor lógico 1. Se existir alguma justificação, os circuitos não são equivalentes. Se existirem implicações que provem que tal assinalamento é inconsistente, os circuitos são equivalentes. Se não existirem nem uma justificação e nem uma prova de algum assinalamento inconsistente para a saída do *miter*, é necessário repetir o primeiro passo, incrementando o nível de recursão.

A maior limitação desse procedimento é que a complexidade de tempo cresce exponencialmente em função do nível de recursão, sendo intratável mesmo para níveis de recursão pequenos (acima de 4). A grande vantagem é que se os circuitos são similares, o nível de recursão é pequeno. Caso contrário, o RL não deve ser utilizado como motor único de resolução.

3.7 CEC Utilizando Resolvedor de SAT

Apesar de serem amplamente conhecidos desde o surgimento do problema de CEC, os resolvedores de SAT foram ignorados por muitos anos pela comunidade de CEC devido à ineficiência dos mesmos. Para resolver uma instância de um problema de CEC entre dois circuitos utilizando um resolvedor de SAT, dois passos iniciais são necessários: a criação de um *miter* e a transformação das portas lógicas em cláusulas em CNF. Com o circuito transformado em uma fórmula com cláusulas em CNF, que é a entrada padrão da maior parte dos resolvedores de SAT atuais, basta criar uma propriedade que obrigue a saída do miter ser 1. Se o resolvedor encontrar algum assinalamento para a fórmula com a propriedade, esse será um contra-exemplo que prova que os circuitos não são equivalentes; se o resolvedor não encontrar tal assinalamento, os circuitos são equivalentes.

A partir da publicação de Larrabee [Lar92], houve um retorno no interesse em se aplicar resolvedores de SAT nos problemas de ATPG existentes na indústria de circuitos integrados. Em relação ao problema de CEC, a maior motivação originou-se dos grandes avanços obtidos nos resolvedores de SAT, principalmente após a publicação do resolvedor de SAT denominado GRASP [SS96].

Entretanto, as técnicas para CEC utilizando resolvedores de SAT ainda eram bastante ineficientes. A primeira metodologia que obteve resultados um pouco melhores quando comparada com os obtidos com BDDs e ATPG foi a de Silva e Silva [SS99]. Como já era clara a ineficiência dos métodos para resolver instâncias de problemas de CEC baseados em resolvedores de SAT, Silva e Silva [SS99] propuseram duas importantes melhorias: a utilização de RL integrado ao resolvedor de SAT e o armazenamento da estrutura topológica do circuito, evitando que a mesma fosse perdida durante a transformação para CNF.

O RL foi utilizado de duas formas para aumentar o desempenho do resolvedor de SAT. A primeira consistiu em fazer um pré-processamento[2] seja na fórmula em CNF que descreve o circuito, ou seja, nas portas lógicas do próprio circuito *miter*. Esse pré-processamento gerava muitas cláusulas de implicações através de RL e as mesmas eram concatenadas à fórmula do problema original com o intuito de reduzir o número de retrocessos e, conseqüentemente, o tempo para resolver a instância. A segunda forma consistiu em integrar o procedimento de RL dentro dos motores de dedução e decisão do resolvedor de SAT. Apesar de ter obtido bons resultados com ambas as formas comparando-se com as outras técnicas baseadas em resolvedores de SAT, a metodologia não foi capaz de superar os resultados obtidos com motores de CEC baseados em BDD e ATPG.

O trabalho de Goldberg et al. [GPB01] foi o primeiro a demonstrar, através de resultados experimentais, que os motores baseados em resolvedores de SAT poderiam obter resultados tão bons quanto os resultados das metodologias baseadas no motor BDD. Os autores estenderam as técnicas de cortes utilizadas nas metodologias com motor BDD e aplicaram em uma metodologia com um motor SAT. Em detrimento de se construir um BDD para o corte, a ferramenta proposta utiliza um resolvedor de SAT simples, baseado em DPLL, com o intuito de provar a equivalência entre os circuitos. Se um resultado negativo surgir, isto é um indicativo de que ambos os circuitos podem não ser equivalentes, um resolvedor SAT mais robusto (GRASP de [SS96]) é utilizado para provar se o resultado era um falso negativo ou um negativo verdadeiro.

As maiores contribuições desse trabalho foram a demonstração de que metodologias para CEC utilizando o motor SAT poderiam ser tão eficientes quanto as metodologias que utilizam o motor BDD; e a proposta de uma heurística de cortes que funciona bem

[2]Devido a importância das técnicas de pré-processamento para SAT, todo o Capítulo 4 foi dedicado ao seu estudo.

com resolvedores de SAT[3].

3.8 CEC Utilizando Motores Híbridos

Paralelamente ao desenvolvimento de motores de resolução únicos para o problema de CEC, alguns autores decidiram integrar a alguns desses motores em apenas uma ferramenta obtendo motores que podem ser classificados como híbridos. A primeira publicação que propôs um motor híbrido é a de Mukherjee et al. [MJT+97]. A motivação de tal trabalho surgiu a partir da observação de que os motores de resolução únicos como ATPG, BDDs, RL e SAT funcionavam bem para instâncias distintas. Por isso, a integração de todos esses motores em apenas um poderia resultar em um grande aumento em desempenho.

A maior dificuldade na proposição de uma ferramenta dessas está na classificação do circuito submetido a CEC em relação a qual tipo de motor que deve ser utilizado. Para isso, os autores sugeriram a utilização de um núcleo de classificação com um sistema denominado *filtering* (filtragem). No entanto, esse núcleo somente funcionava bem para certas classes de circuitos.

Com o intuito de solucionar essas limitações e, principalmente, distribuir as responsabilidades do núcleo de classificação, Burch e Singhal [BS98] propuseram um motor híbrido muito mais integrado. A proposta consiste em classificar o circuito e submetê-lo a um motor único, assim como na proposta anterior. No entanto, o motor único pode abortar o processo de verificação e passar toda a informação obtida na resolução do problema para outro motor. Dessa forma, utilizando motores de maneira intercalada, foi possível obter melhores resultados do que os alcançados com motores únicos e com os motores híbridos existentes.

Desde então, a maior parte das publicações concentrou-se em heurísticas para integrar em um motor híbrido os motores SAT e BDD [RWS00] [KGP01] [DK03]. Atualmente, devido ao grande sucesso dos motores híbridos, a maior parte das ferramentas comerciais de CEC emprega essa abordagem. Entretanto, a pesquisa em motores únicos continua bastante ativa, já que, para se construir ferramentas baseadas em motores híbridos, motores únicos têm que ser utilizados.

[3]Além dos trabalhos citados, outras abordagens importantes também já foram publicadas [AOF07] [MCBE06] [LWC+03].

Capítulo 4

Pré-Processamento para Verificação de Equivalência Baseado em SAT[1]

4.1 Introdução

Esta seção tem como objetivo apresentar uma técnica de pré-processamento que deriva informações redundantes de circuitos sob Verificação de Equivalência de Circuitos Combinacionais (CEC, do Inglês, *Combinational Equivalence Checking*) de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade (SAT) para a prova de equivalência entre ambos circuitos. Através dessa técnica, é possível superar em desempenho as principais técnicas do estado da arte de CEC baseado em SAT.

Como apresentado no Capítulo 3, desde a publicação do trabalho de Goldberg et al. [GPB01], houve um grande interesse na utilização de resolvedores de SAT em problemas de CEC. Outra grande motivação surgiu devido às grandes melhorias feitas nos resolvedores de SAT, principalmente, na última década. Além disso, com o intuito de aumentar ainda mais o desempenho dos resolvedores de SAT, diversas técnicas de pré-processamento de instâncias têm sido propostas.

As técnicas de pré-processamento para instâncias de SAT geralmente aplicam uma das duas técnicas seguintes: a eliminação de variáveis e/ou cláusulas, ou a adição de cláusulas redundantes. A primeira baseia-se no princípio que afirma que quanto menor a instância, menor o tempo para a sua resolução. Em contrapartida, a segunda baseia-se no princípio que afirma que informações redundantes auxiliam a guiar melhor o procedimento de busca do resolvedor de SAT. Ambas as técnicas provaram-se bem sucedidas em diversos tipos de instâncias como detalhado na próxima seção.

[1]Este capítulo da tese resultou em uma publicação de um artigo international [ASF08b].

4.2 Trabalhos Relacionados

As técnicas de pré-processamento tornaram-se um estágio muito importante na verificação formal de hardware utilizando-se resolvedores de SAT. Por isso, diversas abordagens foram propostas explorando essas técnicas. Algumas delas fazem o pré-processamento de fórmulas em Forma Normal Conjuntiva (CNF, do Inglês, *Conjunctive Normal Form*) [LA97][Sil00][BW03][SP04] enquanto outras pré-processam o circuito antes que o mesmo seja convertido para CNF [ASF08b][AH04].

Li e Anbulagan [LA97] trataram a questão do pré-processamento em fórmulas em CNF utilizando técnicas bem interessantes. Eles propuseram uma ferramenta de pré-processamento que é parte do resolvedor de SAT de fórmulas com até 3 literais denominado Satz o qual eles mesmos implementaram. A ferramenta de pré-processamento deles é denominada Compactor e essa ferramenta opera em fórmulas em CNF aplicando três técnicas: a eliminação de literais puros [DP60], a determinação de literais falhados e a adição de resolventes. A primeira técnica consiste em remover as variáveis que aparecem em apenas uma fase, seja ela positiva ou negativa, assim como a remoção das cláusulas que contêm essas variáveis, já que as mesmas são trivialmente satisfeitas. A segunda técnica faz um assinalamento a uma variável e verifica se tal assinalamento resulta em algum conflito. A terceira técnica utiliza uma forma restrita de Resolução aplicada a duas cláusulas (resolução binária) que é capaz de derivar novas cláusulas (resolventes), por exemplo: seja uma fórmula em CNF $\varphi = (x_1 \vee \overline{x}_2) \wedge (x_2 \vee x_3 \vee \overline{x}_4)$, aplicando resolução binária nessa fórmula, uma nova cláusula $(x_1 \vee x_3 \vee \overline{x}_4)$ pode ser obtida.

A ferramenta Simplify [Sil00], proposta por Marques-Silva, utiliza outras técnicas para pré-processar as fórmulas em CNF. Essa ferramenta, que é suportada pela técnica de simplificação algébrica, utiliza duas abordagens: a detecção de equivalência entre duas variáveis da fórmula e a inferência de cláusulas binárias. A primeira abordagem encontra a equivalência entre duas variáveis através de uma técnica de casamento de padrões; sempre que uma variável equivalente a outra é encontrada, uma é substituída pela outra na fórmula. Essa técnica reduz o tamanho da fórmula já que variáveis são eliminadas. A segunda abordagem infere cláusulas através da técnica de casamento de padrões. Marques-Silva observou que certos conjuntos de cláusulas binárias e ternárias sempre resultam nas mesmas cláusulas em CNF aplicando-se a operação de resolução. Por essa razão, certos passos de resolução podem ser substituídos por passos de casamento de padrões.

Recentemente, duas outras importantes ferramentas de pré-processamento, que se provaram bem sucedidas em verificação formal de hardware, foram propostas: a Hypre [BW03] e a NiVER [SP04]. A ferramenta Hypre implementa uma regra de inferência que envolve mais de duas cláusulas por vez, essa técnica é de Hiper-Resolução (do

Inglês, *Hyper-Resolution*). A principal idéia dessa técnica é evitar que cláusulas de tamanho intermediário sejam geradas durante o processo de resolução e que apenas as cláusulas binárias finais sejam utilizadas. Além disso, essa técnica pode produzir cláusulas unitárias simplificando ainda mais a fórmula original. Além disso, a ferramenta Hypre detecta equivalência entre variáveis, o que possibilita uma redução ainda maior da fórmula original.

A ferramenta NiVER implementa um caso especial da Resolução por Eliminação de Variáveis (VER, do Inglês, *Variable Elimination Resolution*), a generalização da resolução binária. O procedimento VER tem complexidade de espaço exponencial. Esse procedimento agrupa dois conjuntos de cláusulas para eliminar uma variável qualquer x; um dos conjuntos deve ter a variável x enquanto o outro deve ter a variável $\bar{x}$. VER aplica a resolução binária a todos os pares de cláusulas de ambos conjuntos, o que resulta em um novo conjunto de cláusulas que não contém a variável x e também resulta na eliminação de outras cláusulas. A ferramenta NiVER utiliza o mesmo procedimento, exceto que a mesma não permite que a fórmula original em CNF aumente o número dos seus literais. Por esse motivo, a NiVER tem complexidade de espaço linear.

Considerada uma extensão da NiVER, a ferramenta SatElite [EB05] utiliza três técnicas novas no pré-processamento de fórmulas em CNF: a resolução por *self-subsuming*, a *subsumption* e a eliminação de variáveis pela substituição. Os autores observaram que o procedimento utilizado pela NiVER geralmente resultava em muitas cláusulas *subsumed*[2] que podem ser eliminadas da fórmula em CNF. Por isso, na SatElite, sempre que uma cláusula é adicionada, todas as cláusulas previamente existentes são verificadas para saber se a nova cláusula é *subsumed* em relação às outras. Além dessas técnicas, a SatElite fornece melhoria significativa nas estruturas de dados o que resulta em uma maior eficiência durante o pré-processamento.

As ferramentas de processamento como a Compactor, a Simplify, a Hypre, a NiVER e a SatElite operam em fórmulas em CNF. Tal fato é considerado uma abordagem mais geral de se lidar com o problema, infelizmente, essa é a forma mais difícil quando se trabalha com o problema de CEC.

Outras abordagens concentram-se em características específicas do tipo de problema que gerou a fórmula em CNF. No contexto de CEC, isso significa pré-processar a descrição do circuito antes de convertê-lo para uma fórmula em CNF. Certamente, essa solução não é tão geral quanto a anterior; no entanto, possui algumas vantagens. A vantagem mais significativa é que a estrutura topológica do circuito, que é perdida sempre que uma conversão para CNF é feita, pode ser utilizada pela ferramenta de pré-processamento[3].

[2] A cláusula ω_1 *subsumes* a cláusula ω_2 se $\omega_1 \subseteq \omega_2$.

[3] Assumindo que a transformação para CNF não foi feita de forma trivial como a Transformação de Tseitin [Tse83].

Arora e Hsiao [AH04] propuseram uma ferramenta de implicações, a IMP2C, que foi utilizada para reduzir o tempo de instâncias de CEC. Antes de converter o circuito para uma fórmula em CNF e submetê-la para o resolvedor de SAT, a IMP2C pré-processa a descrição do circuito. Os resultados mostram que a ferramenta IMP2C é capaz de reduzir o tempo de verificação consideravelmente. Essa ferramenta também é capaz de derivar implicações simples, diretas, indiretas e estendidas, nós constantes e assinalamentos impossíveis, assim como nós equivalentes e complementados em um circuito. No entanto, para se obter um bom desempenho, a ferramenta necessita da configuração de um parâmetro que define a porcentagem total de nós constantes a serem identificados no *miter*. Infelizmente, esse parâmetro varia de acordo com o tipo de instância, o que dificulta a utilização da ferramenta.

Além de verificação formal, as técnicas de pré-processamento também são tópicos de estudo bastante ativos na área de Satisfabilidade Proposicional, área onde algumas conclusões importantes já foram apresentadas.

Boufkhad and Roussel [BR00] mostraram, após executar diversos experimentos com fórmulas aleatórias redundantes e irredundantes de instâncias de SAT, que grande parte das instâncias torna-se mais difícil para o resolvedor de SAT após o tamanho da fórmula ser reduzido por meio da eliminação das cláusulas redundantes. Isso acontece tanto para resolvedores de SAT baseados em procedimentos de busca local bem como para os resolvedores baseados em procedimentos de prova como os derivados do DPLL (Davis, Putnam, Logemann e Loveland).

Ao procurar por uma nova métrica de dificuldade para as fórmulas em CNF em instâncias de SAT, Zeng e McIlraith [ZM06] observaram que instâncias com mais cláusulas redundantes são mais fáceis de serem resolvidas. Além disso, os autores notaram que, ao se retirar todas as cláusulas redundantes de algumas instâncias, as mesmas necessitam de 100 vezes mais tempo de resolução do que a instância original com todas as cláusulas redundantes.

Uma outra conclusão importante foi apresentada por Fourdrinoy et al. [FGMS07] ao estudar o papel da redundância em instâncias de SAT. Os resultados e as conclusões foram estabelecidos após estudar e avaliar o desempenho dos resolvedores de SAT em instâncias que tiveram removidas todas as suas cláusulas redundantes por meio de propagação unitária (UP-redundantes, *Unit Propagation redundant*). Nos experimentos dos autores, eles utilizaram quatro heurísticas para simplificação: a remoção de todas as cláusulas UP-redundantes, a remoção das cláusulas binárias UP-redundantes, a remoção das cláusulas de Horn UP-redundantes, e a remoção de cláusulas binárias e de Horn UP-redundantes. Como as quatro heurísticas reduziam o tamanho da fórmula era esperado que o tempo de resolução também fosse reduzido. No entanto, na maior parte das instâncias, quando as cláusulas binárias e/ou as cláusulas de Horn eram removidas da instância original, o tempo de resolução aumentava consideravelmente.

Como o pré-processamento tornou-se um passo essencial para o aumento do desempenho das técnicas de verificação formal baseadas em resolvedores de SAT, a seção seguinte apresenta a técnica proposta no presente trabalho.

4.3 A Nova Técnica de Pré-Processamento Proposta

Esta seção detalha a técnica de pré-processamento que deriva informações redundantes dos dois circuitos sob CEC de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade para a prova de equivalência entre ambos circuitos. Para fazer a Verificação de Equivalência utilizando essa técnica, os seguintes passos devem ser executados. Primeiro, um circuito *miter* para os dois circuitos nos quais a equivalência deve ser provada deve ser criado, passo necessário para todas as técnicas de CEC baseado em SAT. Segundo, a ferramenta de pré-processamento denominada Vimplic deve percorrer a descrição do circuito *miter* gerando cláusulas binárias (implicações ou informações redundantes) como explicado na Seção 2.3. Terceiro, o circuito *miter* é convertido para uma fórmula em CNF. Essa fórmula é concatenada à propriedade que garante a equivalência entre os circuitos *miter* e também é concatenada à conjunção de cláusulas que foi derivada pela Vimplic; o que resulta em uma nova fórmula em CNF. Quarto e último, essa nova fórmula é submetida ao resolvedor de SAT para que o mesmo prove a equivalência entre os circuitos *miter*.

Para executar os passos descritos anteriormente, a técnica de pré-processamento para Verificação de Equivalência apresentada nessa seção utiliza quatro ferramentas. A primeira é uma ferramenta que cria o circuito *miter* a partir de dois circuitos, converte o *miter* para uma fórmula em CNF e adiciona a propriedade. A segunda é a ferramenta de pré-processamento proposta. A terceira é uma ferramenta muito simples que apenas concatena as cláusulas do circuito *miter* à conjunção de cláusulas produzidas pela Vimplic. A última ferramenta é um resolvedor de SAT como, por exemplo, o Berkmin561 [GN02] ou o Minisat [ES03].

A primeira e a terceira ferramentas são muito simples enquanto a quarta ferramenta, embora não seja tão simples, é disponibilizada em diferentes implementações. A segunda ferramenta, que é a ferramenta que fornece suporte à técnica de pré-processamento apresentada nesse trabalho, é descrita em detalhes no Algoritmo 2.

O Algoritmo 2 representa o algoritmo implementado na ferramenta Vimplic e executa os seguintes passos. Primeiro, a ferramenta lê a descrição do circuito *miter* e cria duas estruturas básicas: um grafo de implicações e uma tabela de (Assinalamento)-(Conjunto Completo de Justificações), ambas vazias. O grafo de implicações é um grafo como explicado na Seção 2.4. A tabela é uma estrutura de dados que armazena em cada uma de suas entradas um assinalamento na saída de uma porta lógica que torna essa porta não-justificada (ver definição na Seção 2.3). Juntamente com esse assi-

nalamento, o conjunto completo de justificações que torna a porta lógica justificada também é armazenado (ver definição na Seção 2.3).

No segundo passo, cada porta lógica lida do arquivo é transformada em um conjunto de vértices e arestas; esse conjunto é adicionado ao grafo de implicações (conforme explicado na Seção 2.4). Enquanto essa transformação é feita, a tabela (Assinalamento)-(Conjunto Completo de Justificações) é preenchida. Apenas as portas lógicas BUFFER e NOT são ignoradas porque essas portas não podem se tornar não-justificadas.

No terceiro passo, uma Busca em Largura é executada na estrutura topológica do circuito *miter*. Sempre que uma porta lógica, com exceção das portas BUFFER e NOT, é encontrada, o par de (Assinalamento)-(Conjunto Completo de Justificações) para essa porta lógica é buscado da tabela. O par é usado para derivar implicações indiretas para a porta lógica encontrada. Relembrado a Seção 2.3, implicações indiretas são geradas pela derivação de implicações diretas para cada justificação do conjunto completo de justificações; e executando, em seguida, a operação de interseção entre os assinalamentos resultantes das implicações diretas. Isso é feito no quarto passo do Algoritmo 2.

O quinto passo consiste em aplicar sucessivamente o procedimento de derivação de implicações simples em vários níveis de portas lógicas de modo a derivar implicações diretas. Finalmente, no sexto e último passo, as implicações simples são derivadas por uma Busca em Largura com o procedimento de ativação (ver definição na Seção 2.4).

Após a descrição do algoritmo, podem ser destacadas as principais diferenças entre a técnica proposta e o trabalho de Arora e Hsiao [AH04]:

- A técnica proposta utiliza o mecanismo de justificação para a derivação de implicações enquanto o mecanismo utilizado por Arora e Hsiao não é claramente descrito no seu artigo [AH04].

- Para derivar as implicações, a técnica proposta utiliza o grafo de implicações parcial, já Arora e Hsiao não explicam qual tipo de grafo de implicações é utilizado.

- A presente técnica utiliza apenas as implicações indiretas para aumentar a eficiência do resolvedor de SAT, já a técnica de Arora e Hsiao utiliza implicações diretas, indiretas e estendidas. No entanto, as definições utilizadas no artigo desses autores não condizem com as definições originalmente publicadas por Kunz and Pradhan [KP94].

- A técnica proposta não utiliza parâmetros específicos dos circuitos e, além disso, a mesma foi provada correta na derivação de implicações. A técnica de Arora e Hsiao não possui nenhuma dessas duas características.

Algoritmo 2: Algoritmo de pré-processamento implementado na Vimplic

Vimplic () Início

// Primeiro Passo
Lê o circuito *miter* no formato BENCH de um arquivo.
Cria um grafo de implicações parcial ou completo vazio de acordo com o parâmetro em linha de comando.
Cria uma tabela de (Assinalamento)-(Conjunto Completo de Justificações) vazia - Tabela ACCJ.

// Segundo Passo
Enquanto (*Não chegar ao fim do arquivo*) **Início**
Lê porta lógica do arquivo.
Converte a porta lógica para a sua representação no grafo de implicações.
Se (*A porta lógica não for uma porta BUFFER ou uma NOT*) **Início**
Armazena o assinalamento a saída da porta lógica atual juntamente com o conjunto completo de justificações na tabela ACCJ.
Fim
Fim

// Terceiro Passo
Executa a Busca em Largura no circuito *miter*.
Para Cada (*Saída de porta lógica encontrada*) **Início**
Obtenha o assinalamento de saída a dessa saída de porta lógica da tabela ACCJ.
Obtenha também o conjunto completo de justificações A_{csj} para esse assinalamento.
DerivaImplicaçõesIndiretas(a**,** A_{csj} **);**
Fim
Fim

// Quarto Passo
DerivaImplicaçõesIndiretas (Assinalamento a**, Conjunto Completo de Justificações** A_{csj} **) Início**
Para Cada (*Assinalamento* b *da* A_{csj}) **Início**
DerivaImplicaçõesDiretas (b **);**
Armazena o conjunto de assinalamentos encontrados.
Fim
Faz a operação de interseção entre os elementos do conjunto direto de implicações encontrado.
Para Cada (*Assinalamento c obtido da operação de interseção*) **Início**
Imprime a implicação $b \rightarrow c$.
Fim
Fim

// Quinto Passo
DerivaImplicaçãoDireta (Assinalamento b **) Início**
Enquanto (*A lista de assinalamentos não estiver vazia*) **Início**
DerivaImplicaçõesSimples(Assinalamento b **)** ;
Adiciona um novo assinalamento à lista de novos assinalamentos.
Fim
Fim

// Sexto Passo
DerivaImplicaçõesSimples (Assinalamento b **) Início**
Executa a Busca em Largura sobre o grafo de implicações a partir do nó inicial b.
Armazena a lista dos nós alcançados.
Fim

4.4 Análise da Técnica de Pré-Processamento Proposta

4.4.1 Introdução

Como detalhado anteriormente, a Vimplic é uma ferramenta de pré-processamento que deriva informações redundantes por meio de implicações simples, diretas e indiretas utilizando a descrição do circuito *miter*. Para se elaborar uma ferramenta como essa, diversas implementações diferentes podem ser feitas. Nessa seção são apresentadas observações experimentais e teóricas a respeito da ferramenta, que auxiliam o entendimento da heurística implementada na mesma.

4.4.2 Determinação de Quais Cláusulas Derivar e Armazenar

Embora a derivação de implicações simples, diretas, indiretas e estendidas seja simples, o número total de implicações a serem derivadas em um circuito é exponencial em relação ao número de variáveis. Por isso, heurísticas devem ser encontradas para determinar o melhor conjunto de cláusulas para se derivar e armazenar.

Implicações simples e implicações diretas derivadas da descrição de um circuito por meio de uma ferramenta como a Vimplic também podem ser diretamente derivadas através de simples aplicações de Propagação de Restrição Booleana (BCP, do Inglês, *Boolean Constraint Propagation*), após a conversão do circuito para CNF. Embora essas implicações evitem que o BCP faça alguns assinalamentos durante a resolução da instância do problema de CEC resultando em uma economia de tempo, o número de implicações (cláusulas) derivadas pela ferramenta pode se tornar extremamente elevado, deteriorando o desempenho do resolvedor de SAT. Além do mais, os resolvedores de SAT atuais geralmente implementam mecanismos de derivação de implicações muito rápidos e eficientes. Por esses motivos, a adição de cláusulas geradas por implicações simples e diretas à fórmula original de um problema de CEC dificilmente resulta em uma redução no tempo de resolução do resolvedor de SAT. É bem mais provável que o excesso de tais cláusulas reduza o desempenho do resolvedor.

Implicações indiretas, no entanto, têm um papel fundamental no aumento de desempenho do resolvedor de SAT, já que essas implicações não podem ser encontradas utilizando-se apenas por BCP. Para encontrar tais implicações, o resolvedor de SAT tem que fazer decisões e ainda aplicar o BCP. Dessa forma, quando implicações indiretas são adicionadas às cláusulas do circuito original descrito por uma fórmula em CNF, tais implicações podam o espaço de soluções evitando muitas decisões e operações de BCP; o que reduz significativamente o tempo de resolução do resolvedor de SAT.

Implicações estendidas são implicações bastante relacionadas com implicações in-

diretas. Arora e Hsiao [AH04] argumentam sobre a grande importância dessas implicações para reduzir o tempo de verificação em problemas de CEC. No entanto, os resultados obtidos no presente trabalho não estão em total acordo com esse argumento. Primeiramente porque, utilizando uma boa parte dos circuitos apresentados no Capítulo 6, duas versões de ferramentas foram testadas. A primeira implementava apenas implicações indiretas e a segunda implementava implicações indiretas e estendidas. Os resultados mostraram que ambas as abordagens são equivalentes em relação ao tempo de verificação.

Além desse resultado experimental, uma observação teórica auxilia a análise de tal argumento. As implicações estendidas podem ser divididas em duas operações: a derivação de uma implicação direta de um nó seguida da derivação de uma implicação indireta, como descrito na Definição 12. Portanto, as implicações estendidas também podem ser derivadas por BCP seguido da derivação de uma implicação indireta. Por esse motivo, como o BCP já é implementado no resolvedor de SAT de forma bastante eficiente, apenas as implicações indiretas são responsáveis por podar o espaço de soluções.

4.4.3 Métodos de Derivação de Implicações Simples, Diretas e Indiretas

A ferramenta Vimplic utiliza uma estrutura de dados simples para a derivação de implicações: o grafo de implicações. Utilizando-se essa estrutura de dados, as implicações simples e diretas podem ser derivadas bastando-se apenas percorrer o grafo de implicações e armazenar cada par de vértices fonte-destino do caminho encontrado, como definido na Seção 2.4. No entanto, implicações indiretas podem ser derivadas utilizando-se no mínimo duas abordagens principais.

A primeira abordagem utiliza um grafo de implicações completo e a lei contra-positiva. Larrabee [Lar92] provou que, se existe um assinalamento em um nó de um circuito descrito por cláusulas em CNF e esse assinalamento satisfaz uma cláusula de 3 literais, então existem implicações que em sua forma contra-positiva resultam em implicações indiretas. Em um grafo de implicações completo, essa situação é representada pela ativação de um vértice AND.

Essa abordagem necessita da aplicação tanto do valor 0 quanto do valor 1 em cada um dos nós do circuito e, cada vez que um vértice AND é ativado no grafo de implicações, a implicação contra-positiva de $a \rightarrow b$ (i.e. $\bar{b} \rightarrow \bar{a}$) gerada por meio desse vértice AND é armazenada; onde a é o vértice inicial e b é o vértice adjacente ao vértice AND.

A segunda abordagem segue exatamente a definição de implicações indiretas apresentada na Seção 2.3 que é baseada no procedimento de justificação. Enquanto a

descrição do circuito é percorrida para a geração do grafo de implicações, os assinalamento e as suas respectivas saídas, que tornam a porta lógica não-justificada, são armazenados juntamente com o conjunto completo de justificações, assim como detalhado na Seção 4.3. Por exemplo, utilizando a Figura 2.3 como referência, pode-se ver que um dos assinalamentos possíveis com o seu conjunto completo de implicações é $f = 0$ juntamente com $J = \{J_1, J_2\}$ onde $J_1 = \{d = 1\}$ e $J_2 = \{e = 1\}$ já que $f = 0$ é o assinalamento que torna a porta lógica NOR não justificada. O par (assinalamento)-(conjunto completo de justificações) é armazenado em uma entrada da tabela, assim como explicado na Seção 4.3.

Após criar essa tabela, cada porta lógica é justificada derivando-se, assim, novas implicações indiretas que são convertidas em cláusulas binárias em CNF.

4.4.4 Grafo de Implicações Completo ou Parcial

Para derivar as implicações simples e diretas tanto o grafo de implicações completo quanto o parcial podem ser utilizados, embora seja evidente que o conjunto de implicações derivado utilizando cada um deles é bem distinto. A partir da Figura 2.6, pode-se observar que metade das arestas e vértices estão relacionados com os vértices AND; os quais só existem em um grafo de implicações completo. Conseqüentemente, removendo-se esses vértices e arestas o número de implicações que pode ser derivado também reduzirá.

A segunda abordagem para derivar implicações explicada na seção anterior é diretamente afetada por essa observação. Como o conjunto de implicações simples e diretas é reduzido, menor é o número de interseções entre o conjunto de assinalamentos das justificações, o que resulta em um decréscimo do número de implicações indiretas possíveis.

4.4.5 Considerações Sobre a Forma de Derivação de Implicações Implementada

Como discutido anteriormente, a derivação de implicações simples e diretas não é complexa e nem exige nenhuma consideração especial. Implicações estendidas não foram implementadas, pois as mesmas não indicaram nenhum ganho em desempenho em relação à utilização de somente implicações indiretas. Por isso, as principais considerações sobre a forma de derivação implementada recaem sobre como derivar implicações indiretas.

A versão inicial da ferramenta Vimplic possuía três métodos para a derivação de implicações indiretas: a derivação de implicações indiretas através da lei contra-positiva (ICP), a derivação através de justificação utilizando o grafo de implicações completo (IJC) e a derivação através de justificação utilizando o grafo de implicações parcial (IJP).

Utilizando um subconjunto dos circuitos apresentados no Capítulo 6, algumas observações experimentais tornaram-se aparentes. O primeiro método (ICP) foi significativamente mais lento do que os outros dois métodos. Isso acontece por diversas razões; primeiro, o número de assinalamentos necessários é extremamente grande já que cada nó do circuito tem que ser assinalado tanto com 0 quanto com 1; segundo, algumas vezes o vértice AND ativado está muito distante do assinalamento inicial no circuito, o que torna o procedimento bastante lento; terceiro, o grafo de implicações completo é quase sempre muito grande, o que torna a busca sobre o mesmo muito lenta.

O segundo método (IJC) é capaz de derivar muito mais cláusulas (implicações) do que o terceiro método (IJP), como esperado. Dependendo do circuito, o número de cláusulas derivadas pode ser extremamente grande, algumas vezes o número de cláusulas chega a ser uma ou duas ordens de grandeza maiores do que o número de cláusulas do circuito original. Isso dificilmente não iria diminuir o desempenho do resolvedor de SAT.

O terceiro método (IJP) deriva um número inferior de cláusulas ao compará-lo com o segundo método, mas utiliza um tempo muito menor. Isso acontece porque o número de vértices do grafo de implicações parcial é consideravelmente reduzido, resultando em uma estrutura de dados bem mais simples que não necessita da representação e da identificação dos vértices AND. Como resultado, a busca sobre o grafo é reduzida drasticamente e, por esse motivo, a versão final da ferramenta Vimplic foi otimizada de modo a derivar implicações indiretas somente através desse método.

4.5 Formalização da Técnica de Pré-processamento de Circuitos

Esta seção apresenta uma garantia de que a técnica de pré-processamento proposta não produz falsos negativos e nem falsos positivos com respeito à equivalência dos circuitos *miter*. Nessa seção, a instância original do problema de satisfabilidade (i.e. a fórmula em CNF do circuito *miter* com a propriedade) é denominada β e a conjunção de cláusulas derivadas pela Vimplic é denominada ρ.

4.5.1 Proposição Geral

Uma forma de se garantir que ρ não altera a satisfabilidade de β é mostrar que $\beta \equiv \beta \wedge \rho$.[4] Para isso, basta mostrar que $\beta \vDash \rho$, pois $\beta \equiv \beta \wedge \rho$ se, e somente se $\beta \vDash \rho$. Por isso, a exatidão da técnica de pré-processamento de circuito pode ser garantida assegurando que as implicações derivadas da Vimplic (ρ) são conseqüências lógicas da fórmula em

[4] β é logicamente equivalente a $\beta \wedge \rho$, i.e. cada assinalamento de variáveis que satisfaz ρ também satisfaz β, e vice-versa

CNF do circuito *miter* (*beta*). Como a ferramenta Vimplic gera implicações simples, diretas e indiretas, cada uma das seções seguintes assegura a exatidão do procedimento utilizado em cada um desses tipos de implicação.

4.5.2 Garantindo Exatidão das Implicações Simples

Implicações simples são derivadas a partir de apenas uma porta lógica de duas entradas. O método para garantir que implicações simples, quando adicionadas à instância original do problema de satisfabilidade, não alteram a sua satisfabilidade é o mesmo para cada uma das portas lógicas. Dessa forma, o procedimento que assegura exatidão será aplicado para apenas uma porta lógica de modo que o mesmo não se torne muito repetitivo. A porta escolhida será uma porta NAND.

Suponha que G seja uma porta lógica NAND com entradas A e B, e saída Z. A porta G pode ser descrita através da sua expressão Booleana $Z = \overline{A \cdot B}$ e pela sua fórmula em CNF:

$$\varphi_G = \underbrace{(Z \vee A)}_{\omega_1} \wedge \underbrace{(Z \vee B)}_{\omega_2} \wedge \underbrace{(\bar{Z} \vee \bar{A} \vee \bar{B})}_{\omega_3}$$

Uma outra forma de se representar as cláusulas ω_1, ω_2 e ω_3 é através de implicações. Essas implicações podem ser divididas em dois tipos: implicações de um literal e implicações de dois literais. Uma implicação de um literal é uma implicação da forma $A \rightarrow B$ enquanto uma implicação de dois literais é uma implicação da forma $(A \wedge B) \rightarrow C$, onde A, B e C são literais. Embora existam outros tipos de implicação, o procedimento de derivação de implicações simples produz apenas implicações de um ou dois literais.

Relembrando-se de ω_1, ω_2 e ω_3, essas cláusulas são logicamente equivalente as seguintes implicações:

$$\omega_1 \equiv (\overline{Z} \rightarrow A) \quad \omega_1 \equiv (\overline{A} \rightarrow Z)$$

$$\omega_2 \equiv (\overline{Z} \rightarrow B) \quad \omega_2 \equiv (\overline{B} \rightarrow Z)$$

$$\omega_3 \equiv ((A \wedge B) \rightarrow \overline{Z}) \quad \omega_3 \equiv (A \wedge Z) \rightarrow \overline{B})$$
$$\omega_3 \equiv ((B \wedge Z) \rightarrow \overline{A})$$

Utilizando o conjunto de implicações recém definido para a porta NAND e o fato que $\varphi_G \vDash \omega_i$ para $1 \leq i \leq 3$ (já que ω_i ocorre em φ_G), uma conclusão direta pode ser mencionada: toda implicação simples derivada pela técnica de pré-processamento

proposta e derivada a partir de uma porta NAND é uma conseqüência lógica da fórmula em CNF da própria porta lógica.

Aplicando-se o mesmo procedimento em outras portas lógicas e nas suas fórmulas em CNF apresentadas, uma conclusão mais geral pode ser estabelecida: toda implicação simples derivada pela Vimplic em qualquer porta lógica é uma conseqüência lógica da fórmula em CNF da própria porta lógica. Dessa forma, implicações simples, quando adicionadas à fórmula original da instância do problema de satisfabilidade, não alteram a satisfabilidade do mesmo.

4.5.3 Garantindo Exatidão das Implicações Diretas

Como definido na Seção 2.3, uma implicação direta consiste na aplicação sucessiva de implicações simples. Suponha que, durante a derivação de implicações diretas, um dos seguintes conjuntos de implicações denominado β' seja encontrado:

$$Z_0 \rightarrow A_0$$

$$A_0 \rightarrow Z_1$$

ou

$$Z_0 \rightarrow A_0$$

$$Z_0 \rightarrow A_1$$

$$(A_0 \wedge A_1) \rightarrow Z_1$$

onde $Z_i \rightarrow A_j$ e $A_m \rightarrow Z_k$ são implicações de um literal e $(A_i \wedge A_j) \rightarrow Z_k$ é uma implicação de dois literais. Essas implicações representam os únicos dois tipos possíveis de implicações produzidas pela Vimplic; e esses dois conjuntos β' são os dois únicos conjuntos de implicações generalizados possíveis. Claramente, para esses dois conjuntos β', $\beta' \vDash Z_0 \rightarrow Z_1$. Assim, $Z_0 \rightarrow Z_1$ é uma implicação direta porque é o resultado de duas ou mais implicações simples (i.e. uma propagação de sinal entre dois ou mais níveis de portas lógicas). Aplicando esse método em diversos níveis de implicações (ou níveis de portas lógicas em um circuito), um novo conjunto de implicações β' pode ser encontrado:

$$Z_0 \rightarrow Z_1$$

$$Z_1 \rightarrow Z_2$$

$$\vdots$$

$$Z_n \rightarrow Z_{n+1}$$

onde $Z_i \rightarrow Z_{i+1}$ é uma implicação direta assim como definido no parágrafo anterior. Por isso, $\beta' \vDash Z_0 \rightarrow Z_{n+1}$, e $Z_0 \rightarrow Z_{n+1}$ é uma implicação direta sobre múltiplos níveis de portas lógicas. Conseqüentemente, por meio das explicações anteriores, uma implicação direta é uma conseqüência lógica de implicações simples.

Relembrando as três considerações seguintes: primeiro, implicações simples são conseqüências lógicas de β; segundo, implicações diretas são conseqüência lógica das implicações simples; terceiro, a conseqüência lógica é uma relação transitiva; pode-se facilmente concluir que implicações diretas, quando adicionadas à instância original do problema de satisfabilidade, não alteram a satisfabilidade do mesmo.

4.5.4 Garantindo Exatidão das Implicações Indiretas

Como definido na Seção 2.3, implicações indiretas são derivadas pela justificação de portas lógicas, o que pode ser formalizado da seguinte maneira: suponha que β' seja o seguinte conjunto de implicações:

$$Z \rightarrow A_0 \vee A_1 \vee \ldots \vee A_n$$

$$A_0 \rightarrow C$$

$$A_1 \rightarrow C$$

$$\vdots$$

$$A_n \rightarrow C$$

Claramente, $\beta' \vDash Z \rightarrow C$ e $Z \rightarrow C$ é uma implicação indireta. Independentemente se β' é composto por implicações simples ou diretas, as implicações indiretas são conseqüências lógicas dessas implicações. Por isso, implicações indiretas, quando adicionadas à instância original do problema de satisfabilidade, não alteram a satisfabilidade do mesmo.

4.5.5 Garantindo Exatidão da Implementação da Vimplic

Na seção anterior, a técnica de derivação de implicações simples, diretas e indiretas foi provada correta. No entanto, isso não é suficiente para garantir que a implementação da ferramenta não possui erros, o que é um fator muito importante no desenvolvimento de novas ferramentas, principalmente na área de verificação formal.

Em verificação formal de hardware, uma ferramenta que produz resultados errados pode levar usuários a análises experimentais incorretas. Em CEC, o problema ainda é mais grave por duas razões. A primeira razão e a mais importante é que usuários podem acreditar erroneamente que dois circuitos são equivalentes quando, na realidade, ambos não são (falso positivo). A segunda razão é que um pesquisador pode implementar

uma ferramenta de implicações para CEC com erros, fazer uma análise experimental e concluir, de forma incorreta, que a sua ferramenta reduz drasticamente o tempo de verificação quando comparada com as ferramentas do estado da arte. Ambas as razões motivam o estudo do problema de garantir exatidão na implementação de ferramentas de implicações.

Existem duas abordagens para lidar com esse problema. A primeira abordagem é provar que o software implementado está correto. Infelizmente, verificação formal de software ainda não é aplicada de forma tão bem sucedida quanto à verificação formal de hardware; além disso, essa abordagem está fora do escopo do presente trabalho. A segunda abordagem é provar que, para uma grande quantidade de circuitos, a ferramenta não deriva nenhuma implicação incorreta.

Embora a última abordagem não garanta a completa exatidão da ferramenta, ela, no mínimo, prova que os resultados mostrados no presente trabalho são corretos. Dessa forma, essa seção apresenta um método simples para provar que as implicações geradas por uma ferramenta de implicações tal como a Vimplic são corretas.

Certamente, existem diversos métodos para alcançar esse objetivo. No entanto, esse trabalho concentra-se na forma mais fácil e mais simples. O presente trabalho utiliza o mesmo método utilizado por Zeng and McIlraith [ZM06] para determinar *Satisfiable Cores* que é baseado na identificação de cláusulas redundantes.

Cláusulas redundantes, como explicado anteriormente, são cláusulas que não adicionam nenhuma informação extra na definição do espaço de soluções de uma fórmula em CNF. Por isso, se essas cláusulas forem removidas da fórmula, o mesmo conjunto de assinalamentos é mantido. A Definição 13 apresenta uma definição formal sobre cláusulas redundantes.

Definição 13 *Seja φ uma fórmula que inclua a cláusula C e seja $\varphi' = \varphi \backslash C$. C é uma cláusula redundante em relação a φ se e somente se $\varphi' \vDash C$.*

Uma importante proposição enunciada por Zeng and McIlraith [ZM06] é que:

Proposição 1 *Dadas uma cláusula C e uma fórmula φ, determinar se C é uma cláusula redundante em relação a φ é um problema tão difícil quando o problema de satisfabilidade.*

De acordo com a Proposição 1, fica claro que não existe nenhum algoritmo realmente eficiente para o problema de determinação de redundância, já que não existe tal algoritmo para SAT. Por isso, o seguinte algoritmo pode ser utilizado sem se preocupar com a eficiência: seja β' uma fórmula em CNF de um circuito *miter* sem a propriedade e seja ρ uma conjunção de cláusulas derivadas pela ferramenta de implicação, então:

1. selecione uma cláusula C da conjunção de cláusulas ρ;

2. resolva a instância do problema de satisfabilidade $\alpha = \beta' \backslash C \cup \neg C$; e

3. se α for insatisfazível, então $\beta' \vDash C$ e C é redundante.

Esse procedimento é repetido para cada cláusula até que a última cláusula de ρ seja testada. Se α for insatisfazível para cada cláusula de ρ, então cada cláusula derivada pela ferramenta de implicação está correta, caso contrário, a ferramenta de implicação está com um erro em sua implementação.

Antes de executar uma análise experimental como apresentado no Capítulo 6, a implementação da Vimplic foi submetida a esse processo de verificação. Como era de se esperar, mesmo fazendo uma implementação cuidadosa dos algoritmos propostos, as primeiras versões da ferramenta Vimplic possuíam um erro na construção do grafo de implicações, o que resultava em implicações incorretas. No entanto, o método apresentado nessa seção foi capaz de detectar a implicação incorreta e o erro foi imediatamente corrigido.

4.6 Resultados

Devido ao grande número de experimentos executados, um capítulo foi destinado para a apresentação e a análise dos mesmos (Capítulo 6). Além disso, grande parte dos resultados disponíveis no Capítulo 6 foram obtidos com auxílio da ferramenta BenCGen que é apresentada no capítulo seguinte (Capítulo 5).

Capítulo 5

Ferramenta para Geração de Circuitos para Benchmarks - BenCGen[1]

O objetivo desse capítulo é apresentar a ferramenta proposta de geração de circuitos para benchmarks. O presente capítulo foi divido em cinco seções. A primeira seção faz uma breve introdução sobre os benchmarks existentes para verificação formal e a segunda seção apresenta uma análise dos benchmarks mais populares bem como um detalhamento das suas principais características. Na terceira seção, uma descrição da ferramenta BenCGen (do Inglês, *BENchmark Circuits GENeration tool*) é feita abordando tanto aspectos de implementação quanto características dos circuitos produzidos. A quarta seção apresenta um comparativo entre os principais resolvedores de Satisfabilidade (SAT) na resolução de instâncias de problemas de Verificação de Equivalência de Circuitos Combinacionais (CEC, do Inglês, *Combinational Equivalence Checking*) por meio de um benchmark produzido pela ferramenta BenCGen. Finalmente, a quinta seção apresenta uma análise dos resultados obtidos na seção anterior.

5.1 Introdução

Benchmarks de circuitos são mecanismos importantes para guiar a escolha de ferramentas para a Automação de Projetos Eletrônicos (EDA, do Inglês, *Electronic Design Automation*). Devido à enorme diversidade de áreas de EDA e a grande variedade de ferramentas comerciais existentes para cada uma dessas áreas, existe uma demanda crescente de novos benchmarks de circuitos. Além disso, como a escala de integração dos circuitos digitais aumentou drasticamente ao longo das últimas décadas, os benchmarks estão se tornando desatualizados e inadequados muito rapidamente.

[1]Este capítulo da tese resultou em uma publicação de um artigo international [ASF08a].

Um benchmark para EDA é uma coleção de circuitos em um formato comum que pode ser utilizado para avaliar algoritmos e ferramentas dado o domínio de um problema [Har00]. Na EDA, os domínios de problemas mais comuns em que benchmarks são utilizados são: Verificação Formal (Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais), simulação de falhas, síntese lógica, mapeamento tecnológico e análise de testabilidade.

Existem duas classes principais nas quais os benchmarks para EDA podem ser divididos e essa divisão é feita de acordo com os tipos de circuitos que os benchmarks possuem. Por um lado, existe a classe dos benchmarks industriais e, por outro lado, existe a classe dos benchmarks sintéticos. Os benchmarks baseados na primeira classe tem sido a escolha mais popular nas últimas décadas. Nessa classe, os benchmarks mais utilizados nas áreas de simulação de falhas, de análise de testabilidade e de verificação de equivalência de Circuitos Combinacionais e Seqüenciais são o ISCAS 85 [BF85] e o ISCAS 89 [FBK89].

Recentemente, dois outros importantes benchmarks tornaram-se disponíveis: o benchmark ITC 99 [Dav99] (do Inglês, *International Test Conference*) e os benchmarks Velev [Vel04]. Os primeiros são utilizados freqüentemente em análise de testabilidade, simulação de falhas e verificação formal enquanto os últimos são utilizados principalmente em verificação formal por meio de resolvedores de Satisfabilidade.

Mesmo sendo a escolha predominante em diversas áreas de EDA, os benchmarks industriais nem sempre são os mais adequados para avaliar ferramentas devido a três principais aspectos. O primeiro aspecto e certamente o mais importante é que as empresas de circuitos integrados não disponibilizam projetos de circuitos atualizados já que os mesmos constituem uma parte importante da propriedade intelectual dessas empresas. Por isso, quase a totalidade dos projetos de circuitos utilizados nos benchmarks industriais são desatualizados. O segundo aspecto é que apenas um conjunto bem limitado de circuitos torna-se público, o que é justificado pela mesma razão apresentada anteriormente: valor de propriedade intelectual. O terceiro aspecto é que, freqüentemente, apenas circuitos com tamanhos e formatos bem específicos são oferecidos.

Por exemplo, o benchmark ISCAS 85 disponibiliza apenas um multiplicador que é da arquitetura *Array* e somente no formato BENCH (ISCAS). Se um algoritmo ou ferramenta proporcionar ótimos resultados nesse circuito não significa que os mesmos manterão a mesma qualidade de resultados para outros multiplicadores *Array* com operandos de 8 ou de 32 bits ou, muito menos, em um multiplicador *Dadda Tree* com operandos de outro tamanho qualquer. Dessa forma, benchmarks industriais sozinhos não são suficientes para fornecer uma grande variedade de circuitos para avaliação de ferramentas comerciais de projeto e verificação de circuitos integrados.

O principal objetivo dos benchmarks sintéticos é superar as limitações dos benchmarks industriais apresentadas anteriormente. Benchmarks sintéticos para EDA são

coleções de circuitos gerados por uma ferramenta a partir de certas características dadas. Uma das primeiras abordagens de sucesso na geração de benchmarks sintéticos foi utilizada na área de análise de roteabilidade de FPGAs (do Inglês, *Field-Programmable Gate Arrays*) [DD96]. Após esse trabalho, diversos outros benchmarks foram propostos [MHC02][PLM99][VSC02].

A maior dificuldade em se utilizar exclusivamente benchmarks sintéticos é que a maioria desses é baseada em ferramentas de geração de circuitos aleatórios, o que resulta em circuitos que são muito diferentes dos utilizados no dia-a-dia. Portanto, em determinados casos, pode ser inútil avaliar uma ferramenta por meio de tais benchmarks.

Uma solução intermediária é a geração sintética de circuitos que são freqüentemente usados na indústria de semicondutores [BAA+01][HWAH06][2]. O maior benefício dessa abordagem é que os circuitos gerados não são tão artificiais quanto aqueles derivados de ferramentas de geração de circuitos aleatórios e, nem tão específicos, limitados, e, na maior parte das vezes, desatualizados como os fornecidos em benchmarks industriais.

Com a finalidade de estender e melhorar as soluções intermediárias existentes, esse capítulo apresenta a ferramenta de geração de circuitos para benchmarks denominada BenCGen.

5.2 Análise dos Benchmarks Mais Populares

Como existem diversos domínios para EDA e ainda muitos benchmarks para cada domínio, essa seção concentra-se nos benchmarks utilizados em verificação formal de hardware já que esse é o escopo do presente trabalho. Assim, esta seção analisa os principais e mais populares benchmarks utilizados em Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais, simulação de falhas e análise de testabilidade.

Um dos primeiros e mais populares benchmarks na área de verificação formal de hardware é o ISCAS 85 [BF85]. Esse benchmark é composto por 10 circuitos combinacionais que implementam funções como unidades lógicas e aritméticas, controladores de interrupção, unidades de correção de erros e unidades de detecção de erros. O número de portas lógicas dos circuitos do ISCAS 85 varia de 160 portas (circuito c432, um controlador de interrupção) a 3512 portas (circuito c7552, um comparador e somador de 32 bits). Devido ao pequeno tamanho e a baixa complexidade desses circuitos, e também devido ao número muito limitado de circuitos, o ISCAS 85 recebeu uma extensão denominada ISCAS 89 [FBK89].

[2]Apesar de esses dois trabalhos implementarem ferramentas de geração sintética de circuitos, o objetivo final de ambos os trabalhos é fomentar o reaproveitamento de projetos de circuitos, reduzindo o custo de novos projetos. Ambos os trabalhos nem citam que os circuitos derivados dos mesmos podem ser utilizados em benchmarks. No entanto, mesmo com grandes limitações, como mostrado na seção seguinte, ambos os trabalhos podem ser utilizados para essa finalidade.

O benchmark ISCAS 89 é uma coleção de 31 circuitos seqüenciais com funções como controladores de semáforos, multiplicadores de números fracionais, controladores de propósito geral e cópias desses mesmos circuitos após a remoção de todas as redundâncias em modo *full-scan*. O tamanho e a complexidade dos circuitos variam de 10 portas lógicas e 3 flip-flops do tipo D (circuito s27) a 22179 portas lógicas e 1636 flip-flops do tipo D (circuito s38417). Esses novos circuitos certamente proporcionam um acréscimo significativo em tamanho e complexidade quando os comparamos com os circuitos do ISCAS 85.

Apesar de os benchmarks ISCAS (ISCAS 85 e ISCAS 89) terem sido utilizados por quase duas décadas, os mesmos estão muito desatualizados. O rápido crescimento da complexidade dos circuitos integrados ao longo dos anos resultou em circuitos com centenas de milhares de portas lógicas. Conseqüentemente, os benchmarks ISCAS não representam mais os circuitos utilizados nas indústrias de semicondutores na última década. Por essa razão, novos benchmarks foram propostos desde a publicação da última versão dos benchmarks ISCAS em 1989.

Com o intuito de superar as sérias limitações dos benchmarks ISCAS, um outro benchmark denominado ITC 99 [Dav99] foi proposto em 1999. O ITC 99 é formado por 31 circuitos: três circuitos fornecidos por indústrias de semicondutores, três disponibilizados por universidades, três versões atualizadas de circuitos previamente existentes no ISCAS 89 e os outros 22 circuitos restantes são conhecidos como benchmarks do Politecnico de Torino. Como a maioria dos circuitos presentes no ITC 99 foi obtida do benchmark do Politecnico de Torino, somente os circuitos pertencentes a esse último benchmark são detalhados.

O benchmark do Politecnico de Torino disponibiliza 16 circuitos novos e 6 circuitos que foram implementados a partir de múltiplas cópias desses circuitos novos. O número de portas lógicas e de flip-flops do tipo D varia de 46 e 5 para o circuito b01 e de 68752 e 3320 para o circuito b18. Certamente, essa é uma grande melhoria no tamanho e na complexidade dos circuitos comparando-se com os circuitos disponibilizados nos benchmarks ISCAS. Embora o benchmark do Politecnico de Torino seja freqüentemente utilizado, esse benchmark é muito limitado no número de circuitos, além disso, não há garantia de que os circuitos disponibilizados estão corretos já que os mesmos foram obtidos de sítios na internet e, ainda, somente circuitos de tamanhos específicos (circuitos não parametrizados) são fornecidos.

Recentemente, uma nova versão de um benchmark denominado benchmark do Workshop Internacional em Lógica e Síntese (do Inglês, *International Workshop on Logic and Synthesis* - IWLS) foi publicada em 2005 [Alb05]. A sua versão inicial foi publicada em 1989 e a sua versão atual (IWLS 2005) é a mais completa reunindo 84 circuitos. Quase a totalidade desses circuitos foi obtida de outros benchmarks como o ITC 99 (21 circuitos), ISCAS (30 circuitos) e OpenCores [Ope08] (26 circuitos). Os dois

primeiros benchmarks já foram analisados anteriormente. Por isso, apenas os circuitos do OpenCores são analisados a seguir.

O benchmark do OpenCores é formado por uma coleção de circuitos digitais livremente disponibilizados no sítio do OpenCores na internet. Qualquer pessoa que quiser divulgar o seu novo circuito digital pode submetê-lo através do sítio e deixá-lo disponível. Infelizmente, não existe nenhum padrão de desenvolvimento ou estilo de projeto que agrupe os circuitos em uma mesma identidade. Além disso, não existe nenhuma garantia de que os circuitos disponibilizados estão livres de erros[3] ou até mesmo que esses circuitos implementam a funcionalidade descrita para eles. Por essas razões, o benchmark do OpenCores não se tornou tão popular quanto os benchmarks do Politecnico de Torino e os ISCAS, mesmo com o conhecimento das diversas limitações desses últimos.

Em 2004, uma nova geração de benchmarks ISCAS, popularmente conhecida como benchmarks Velev [Vel04], foi proposta. Esse benchmark concentra-se em instâncias de circuitos derivados de processadores e a sua aplicação é na área de verificação formal. Em relação ao número de portas lógicas, o maior circuito disponibilizado supera em uma ordem de grandeza maior o maior circuito do benchmark do Politecnico de Torino, tal circuito possui mais de 700.000 portas lógicas. Os circuitos que o compõem são derivados de microprocessadores incluindo instâncias de processadores com pipeline, super-escalares, multi-ciclo, unidades funcionais, modelos de Palavra de Instrução Muito Longa (VLIW, do Inglês, *Very-Long Instruction Word*), dentre outros.

A principal limitação dos benchmarks Velev é que o mesmo fornece arquivos de descrição do circuito, tal como o BENCH (ISCAS) ou o BLIF (do Inglês, *Berkeley Logic Interchange Format*), para apenas 4 dos 28 benchmarks existentes. Para todos os outros benchmarks, apenas a fórmula em CNF é disponibilizada. Dessa forma, diversas áreas como a de Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais, a de simulação de falhas e a de análise de testabilidade não podem se beneficiar desses benchmarks. No entanto, os benchmarks Velev são uma contribuição importante para o desenvolvimento de tópicos de pesquisa como ordenação de variáveis para problemas de SAT e a aplicação de técnicas de pré-processamento para simplificação de fórmulas.

Além dos benchmarks previamente descritos, ferramentas de geração de circuitos podem ser utilizadas para produzir benchmarks sintéticos. As duas ferramentas mais populares nessa área são: o Gerador de Módulo Aritmético [HWAH06] (AMG, do Inglês, *Arithmetic Module Generator* e a Eudoxus [BAA+01]. Ambas as ferramentas foram projetadas para possibilitar a reutilização de códigos, por isso, essas ferramentas só produzem circuitos em Nível de Transferência de Registros (RTL, do Inglês, *Register Transfer Level*), o que é inadequado para muitas áreas como a de Verificação de Equivalência de Circuitos Combinacionais e Seqüenciais, a de simulação de falhas e

[3]Certamente, a Verificação de Equivalência pode ser feita com duas cópias de um circuito com erros, no entanto, isso raramente acontece.

a de análise de testabilidade. Além disso, essas ferramentas não produzem circuitos em nenhum dos formatos populares de descrição de circuitos para verificação formal como o BENCH, o BLIF ou EQN. Um outro fator importante é que essas ferramentas não implementam os blocos básicos essenciais que são utilizados em quase todos os circuitos como: (de)multipliexadores, (de)codificacores, dentre outros. Finalmente, o número de portas dos circuitos gerados por essas ferramentas é limitado.

Após descrever os principais benchmarks para as áreas de Automação de Projetos Eletrônicos (EDA, do Inglês, *Electronic Design Automation*), algumas observações podem ser feitas. Primeiro, o esforço de disponibilizar benchmarks de circuitos deve ser contínuo ao longo dos anos, já que a escala de integração e a complexidade dos circuitos integrados tem crescido continua e rapidamente ao longo das últimas décadas. Segundo, nem os benchmarks industriais e nem os benchmarks sintéticos devem ser utilizados separadamente para avaliar novas ferramentas. Certamente, esses dois tipos de benchmarks não são competidores, mas, sim, complementares. Terceiro, circuitos de tamanho parametrizados e circuitos com a mesma função e diferentes arquiteturas devem ser disponibilizados para fornecer uma análise mais segura de ferramentas comerciais em relação a uma classe específica de circuitos. Por esses motivos, a seção seguinte apresenta a ferramenta BenCGgen.

5.3 Geração Sintética de Benchmarks - BenCGen

5.3.1 Introdução

BenCGen é uma ferramenta de geração de circuitos digitais para benchmarks. Essa ferramenta produz circuitos que podem ser utilizados em diversas áreas como na Verificação de Equivalência de Circuitos Combinacionais, na simulação de falhas e na análise de testabilidade. A ferramenta BenCGen gera uma grande variedade de circuitos combinacionais, principalmente, os aritméticos e circuitos com funções básicas tais circuitos como (de)multiplexadores, (de)codificadores, comparadores, dentre outros. Esses circuitos podem ser divididos em sete classes como mostra a Tabela 5.1.

A primeira classe apresentada na Tabela 5.1 é a de circuitos multiplicadores. Os multiplicadores receberam uma atenção especial na ferramenta BenCGen pois os mesmos são considerados a classe de circuitos combinacionais mais difícil para ser verificada formalmente. A BenCGen implementa as arquiteturas de multiplicadores mais importantes e mais empregadas na indústria de semicondutores: multiplicadores *Array*, *Carry LookAhead*, *Reduced Tree*, *Dadda Tree* e *Wallace Tree*. Os circuitos multiplicadores gerados sempre possuem operandos com o mesmo número de bits.

A segunda classe de circuitos, os somadores, são os blocos básicos mais importantes de qualquer circuito aritmético. Devido ao seu amplo número de aplicações, todos os

Tabela 5.1: Características dos circuitos que podem ser gerados pela ferramenta BenCGen

#	Nome do Circuito	Classe	Número de Bits	Número de Portas
1	*Array*	Multiplicador	704	2.852.412
2	*Carry LookAhead*	Multiplicador	512	4.258.277
3	*Reduced Tree*	Multiplicador	704	3.913.760
4	*Dadda Tree*	Multiplicador	704	3.913.715
5	*Wallace Tree*	Multiplicador	704	4.245.894
6	*Ripple Carry*	Somador	102.400	512.002
7	*Carry LookAhead*	Somador	1408	990.521
8	*Block Carry LookAhead*	Somador	102.400	665.603
9	*Carry Save*	Somador	102.400	512.003
10	*Carry Skip*	Somador	102.400	716.803
11	*Carry Select*	Somador	102.400	1.945.576
12	*Ripple Carry*	Somador Subtrator	102.400	614.404
13	*Barrel Shifter*	Shifter	16	4.718.597
14	*Non Restoring*	Divisor	1280	9.831.682
15	*Restoring*	Divisor	1024	14.684.162
16	*Carry LookAhead*	Divisor	704	4.956.867
17	Multiplexador	Básico	131.072	262.164
18	Demultiplexador	Básico	131.072	262.163
19	Decodificador Binário	Básico	131.072	131.091
20	Codificador Binário	Básico	131.072	131.091
21	Codificador de Prioridade	Básico	131.072	1.048.832
22	Comparador de Magnitude	Básico	102.400	400.020
23	Circuito de Paridade	Básico	131.072	102.401
24	Unidade Lógica e Aritmética	Aritmético	512	2.500.362

principais tipos de somadores foram implementados: somadores *Ripple Carry*, *Carry LookAhead*, *Block Carry LookAhead*, *Carry Save*, *Carry Select* e *Carry Skip*. Além disso, também foi implementado um somador-subtrator *Ripple Carry*.

Uma outra classe de circuitos muito importante é a classe dos divisores. Embora alguns projetos atuais utilizem divisores seqüenciais, os mesmos são geralmente muito lentos para processamento de multimídia e de dados em tempo real. Por isso, nessas áreas, a escolha mais comum é a de divisores combinacionais tais como o divisor *Restoring*, o *Non-Restoring* e o *Carry LookAhead*. Todos esses três tipos de divisores podem ser gerados pela BenCGen.

Outra classe de circuitos bastante relevante é a que implementa os circuitos combinacionais básicos. Embora sejam circuitos muito simples, os mesmos são instanciados

em quase todos os projetos de circuitos mais complexos. As funções básicas de circuitos que podem ser produzidas são os codificadores, decodificadores, multiplexadores, demultiplexadores, codificadores de prioridade, comparador de magnitude e circuito de paridade.

Finalmente, para que a ferramenta fornecesse um conjunto bastante completo de circuito combinacionais, uma função responsável pela geração de Unidades Lógicas e Aritméticas (ULAs) foi implementada. Essa unidade tem 16 operações básicas: adição, subtração, multiplicação, divisão, deslocamento para direita, deslocamento para esquerda, rotacionamento para esquerda, rotacionamento para direita, incremento, decremento, identidade, zero e as operações lógicas AND, OR, XOR e NOT.

5.3.2 Detalhes de Implementação

A BenCGen foi implementada em linguagem C++ explorando o paradigma de orientação a Objetos de modo a reduzir o esforço de geração da ferramenta. Circuitos digitais, tais como produzidos pela BenCGen, são bastante hierárquicos e quase todos eles são compostos por blocos básicos. A Orientação a Objetos foi utilizada de modo a explorar bastante essas duas características.

As duas questões principais ao se projetar uma ferramenta de geração de circuitos são: a implementação parametrizada de blocos básicos e a conexão entre esses blocos de forma a produzir circuitos maiores.

A implementação dos blocos básicos é determinada principalmente pelo nível de abstração do circuito a ser gerado. Como a BenCGen produz circuitos em nível de portas lógicas e em RTL, alguns blocos foram implementados duas vezes, uma para cada nível de abstração. Por exemplo, enquanto um somador completo é formado apenas por um conjunto de portas lógicas em uma descrição em nível de portas, em RTL, o mesmo somador é representado por um módulo em Verilog. Dessa forma, os blocos básicos para uma descrição em nível de portas lógicas são as próprias portas; e os blocos básicos em RTL são os meio-somadores, os somadores completos, os (de)codificadores, os (de)multiplexadores, dentre outros.

Depois de implementar os blocos básicos em ambos níveis de abstração, blocos maiores como os multiplicadores e os divisores foram implementados facilmente por meio de reuso dos blocos básicos já existentes. A conexão entre blocos foi um fator muito relevante durante a confecção da BenCGen. Com o intuito de tornar o procedimento de conexão entre os fios dos blocos mais eficiente e mais simples, determinou-se que cada bloco fosse responsável apenas pela criação dos seus blocos internos e que os mesmos fossem instanciados de uma forma *top-down*.

Por exemplo, suponha que um novo objeto do tipo Divisor *Non Restoring* seja criado na ferramenta. Esse objeto é composto por um arranjo bi-dimensional de subtratores

controlados e portas lógicas OR e NOT. Primeiramente, o divisor cria os sinais de entrada e saída para o primeiro subtrator controlado. Em seguida, esse subtrator, que é composto internamente apenas por portas lógicas, cria os sinais e gerenciará as conexões entre suas portas. Esse procedimento continua até que o último bloco do circuito seja criado e conectado aos outros blocos já existentes.

5.3.3 Provando a Exatidão dos Circuitos Gerados

Provar a exatidão dos circuitos gerados pela BenCGen é uma questão essencial, uma vez que esses circuitos serão utilizados para avaliar diversas ferramentas. Se um circuito gerado não executar a sua função devida e ele for utilizado para avaliar uma ferramenta, muitas observações incorretas poderão ser feitas a respeito do desempenho da mesma. Por isso, a maior parte dos circuitos gerados pela ferramenta BenCGen foi provada funcionalmente correta de modo a prover segurança e confiança aos usuários.

A utilização de uma *golden reference*, que é um circuito que já foi provado correto anteriormente, é uma das maneiras mais fáceis de se provar a exatidão de um circuito novo. A prova pode ser feita através da verificação de equivalência entre a *golden reference* e o circuito sob verificação (por exemplo, um circuito gerado pela BenCGen).

Como apresentado no Capítulo 3, a Verificação de Equivalência de Circuitos Combinacionais pode ser feita por meio de diversos motores como Geração Automática de Padrões de Testes (ATPG, do Inglês, *Automatic Test Pattern Generation*), Diagrama de Decisão Binária (BDD, do Inglês, *Binary Decision Diagram*), Diagrama de Momento Binário (BDD, do Inglês, *Binary Moment Diagram*), Aprendizado Recursivo (RL, do Inglês, *Recursive Learning*) e resolvedores de Satisfabilidade (SAT). Como o presente trabalho concentra-se em SAT, a exatidão dos circuitos da BenCGen será provada utilizando esse motor. Relembrando o Capítulo 3, a verificação de equivalência pode ser feita por meio de três passos simples.

Primeiro, um *miter* é construído para que a Verificação de Equivalência de Circuitos Combinacionais através do resolvedor de SAT seja feita. Segundo, o circuito *miter* é convertido para uma fórmula em CNF com uma propriedade que assegura que a variável que representa a saída do *miter* não poderá ser assinalada com 1. Finalmente, essa fórmula em CNF e a propriedade são submetidas ao resolvedor de SAT. O resolvedor tenta provar que existe um assinalamento para a fórmula em CNF que a torne verdadeira (i.e. um contra-exemplo que prova que ambos os circuitos não são equivalentes). Se tal assinalamento não existir, ambos os circuitos são equivalentes.

Após executar esses três passos, o circuito poderá estar correto. No entanto, encontrar uma *golden reference* é uma questão muito difícil. No caso do presente trabalho, esses circuitos de referência foram obtidos do *Altera Quartus II MegaWizard Plug-in Manager* [Alt08] que é um gerador de circuitos digitais verificados funcionalmente.

Como o *Quartus II MegaWizard* não fornece a descrição do circuito em Verilog, a descrição do mesmo foi exportada para o formato BLIF através do gerador de *netlist* do software. A partir daí, um *miter* foi construído utilizando a *golden reference* da Altera e o circuito gerado pela BenCGen.

Na realidade, a ferramenta da Altera não fornece todos os circuitos que podem ser gerados pela BenCGen. Essa ferramenta disponibiliza apenas um pequeno subconjunto. Contudo, a mesma fornece pelo menos um circuito de cada classe existente na BenCGen como um multiplicador, um divisor, um somador e um circuito para cada função básica. Assim foi possível garantir a exatidão dos circuitos produzidos pela BenCGen por meio de CEC.

O único circuito que não foi verificado completamente foi a Unidade Lógica e Aritmética (ULA) devido a dois motivos. Primeiro, não existe uma *golden reference* e nem mesmo um gerador de ULAs com operandos de tamanhos parametrizados foi encontrado. Segundo, mesmo utilizando uma ULA com operandos de tamanho fixo disponível, seria altamente improvável que ela implementasse o mesmo subconjunto de funções implementada na ULA gerada pela BenCGen. Entretanto, as menores instâncias dessas ULAs foram simuladas exaustivamente.

5.4 Resultados de CEC Usando a BenCGen

Essa seção apresenta os primeiros resultados de Verificação de Equivalência de Circuitos Combinacionais utilizando a ferramenta de geração de circuitos para benchmarks, a BenCGen. O principal objetivo dessa seção é prover resultados para guiar futuros pesquisadores na seleção do resolvedor de SAT mais apropriado para o problema de CEC.

Existe uma competição internacional de SAT [wp08] que classifica os três melhores resolvedores de SAT para cada uma de três categorias de instâncias: instâncias industriais, instâncias feitas a mão e instâncias aleatórias. Cada categoria é dividida em três tópicos específicos de acordo com o resultado esperado para a instância; os resultados podem ser SAT, UNSAT ou SAT e UNSAT.

Como as instâncias de CEC são tipicamente industriais e sempre UNSAT (assumindo o pior caso em relação de tempo de resolução), os dois melhores resolvedores apresentados no site da competição internacional de SAT foram selecionados para um comparativo de desempenho, são eles: o Rsat [PD07] e o Minisat [ES03]. Além disso, outros dois importantes resolvedores que não participaram da Competição de SAT do último ano, mas que geralmente oferecem bons resultados, foram selecionados, são eles: o Berkmin561 [GN02] e o Siege_v4 [Rya04]. Todos os resolvedores foram submetidos a um benchmark gerado pela ferramenta BenCGen.

O benchmark gerado pela BenCGen é composto por diversos circuitos de classes

Tabela 5.2: Resultados para CEC de duas cópias de um mesmo multiplicador *Carry LookAhead*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
3x3	326	0,02	0,01	0,01	0,00
4x4	686	0,02	0,01	0,01	0,00
5x5	1242	0,09	0,08	0,08	0,03
6x6	2046	0,60	0,45	0,63	0,42
7x7	3158	4,73	2,52	9,35	5,60
8x8	4646	61,35	34,47	160,82	52,97
9x9	6586	350,11	145,61	1851,65	184,04
10x10	9062	1710,42	708,18	5409,92	496,44
11x11	12166	>10000	>10000	6363,47	606,15
12x12	15998	-	-	>10000	1191,03
13x13	20666	-	-	-	1916,41
14x14	26286	-	-	-	2903,37
15x15	32982	-	-	-	4077,87
16x16	40886	-	-	-	5218,68
17x17	50138	-	-	-	8742,14
18x18	60886	-	-	-	>10000

(Tempos de execução medidos em segundos.)

como multiplicadores, divisores, somadores e funções básicas de circuito. Os resultados foram obtidos utilizando-se um computador com um processador Pentium IV 2,8 GHz com 1,5GB de memória com o sistema operacional Suse Linux 10.0. O tempo de resolução dos resolvedores de SAT foi medido em segundos e o limite de tempo foi configurado para 5,000 ou 10,000 segundos de acordo com a classe do circuito. O resolvedor Berkmin561 foi executado com o parâmetro *s 1* que ativa uma estratégia especial para verificação de equivalência e a semente escolhida para o Siege_v4 a de valor igual a 100.

As Tabelas 5.2 a 5.5 apresentam os resultados dos resolvedores do estado da arte em SAT em problemas de CEC entre cópias redundantes de multiplicadores *Carry LookAhead*, *Wallace Tree*, *Array* e *Dadda Tree* respectivamente. A primeira observação é que o Berkmin561 é capaz de provar a exatidão de instâncias de três a seis vezes maiores em relação ao número de cláusulas do que o Rsat, o Minisat e o Siege_v4. A segunda observação é que, com exceção do Berkmin561, todos os outros três resolvedores de SAT apresentam um crescimento exponencial no tempo de resolução mesmo com pequenos acréscimos no número de cláusulas das instâncias.

A Tabela 5.6 e a Tabela 5.7 apresentam os resultados dos resolvedores do estado da arte em problemas de verificação de equivalência entre cópias redundantes de divisores *Restoring* e *Non Restoring* respectivamente. Nessas instâncias, o Berkmin561 pode ser considerado novamente o resolvedor de SAT mais adequado, já que ele é capaz

Tabela 5.3: Resultados para CEC de duas cópias de um mesmo multiplicador *Wallace*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
3x3	534	0,01	0,00	0,010	0,00
4x4	1060	0,01	0,01	0,01	0,01
5x5	1788	0,07	0,06	0,07	0,04
6x6	3116	0,88	0,73	0,70	0,34
7x7	4392	8,85	4,67	8,39	3,51
8x8	5968	130,60	50,68	125,60	89,60
9x9	7882	1837,68	469,03	2981,50	319,83
10x10	9980	>10000	2456,87	>10000	305,65
11x11	12422	-	>10000	-	634,23
12x12	15254	-	-	-	863,04
13x13	18390	-	-	-	1131,61
14x14	21808	-	-	-	2829,88
15x15	27102	-	-	-	2445,51
16x16	30084	-	-	-	2875,38
17x17	34732	-	-	-	3635,50
18x18	41576	-	-	-	4013,71
19x19	47272	-	-	-	5066,39
20x20	51342	-	-	-	9162,34
21x21	60062	-	-	-	7686,99
22x22	67218	-	-	-	>10000

(Tempos de execução medidos em segundos.)

de resolver instâncias cinco vezes maiores em número de cláusulas do que os resolvedores Rsat e Minisat. Uma outra observação interessante é que o Siege_v4 apresenta resultados ligeiramente melhores do que os obtidos com o Berkmin561 nas instâncias do divisor *Restoring*. Todavia, esses bons resultados não persistiram para os divisores *Non Restoring*. Por essa razão, o Berkmin561 pode ser classificado com o resolvedor de SAT mais robusto e mais adequado para a Verificação de Equivalência de Circuitos Combinacionais dentre os resolvedores selecionados nas instâncias de utilizadas.

Finalmente, a Tabela 5.8 apresenta os resultados dos resolvedores do estado da arte em problemas de verificação de equivalência entre cópias redundantes de somadores *Ripple Carry*. A classificação de desempenho dos resolvedores foi a seguinte: Minisat, Siege_v4, Rsat e Berkmin561. Embora o Berkim561 apresente o maior tempo de resolução, a diferença entre os tempos dele em relação ao Minisat é bem pequena. Além disso, apesar de ter ficado em último, o Berkmin561 não apresenta um crescimento exponencial no tempo em relação ao aumento de cláusulas como aconteceu com os outros resolvedores de SAT nas instâncias anteriores.

Tabela 5.4: Resultados para CEC de duas cópias de um mesmo multiplicador *Array*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
3x3	230	0,00	0,00	0,01	0,00
4x4	466	0,01	0,01	0,01	0,00
5x5	782	0,07	0,07	0,04	0,03
6x6	1178	0,36	0,42	0,04	0,25
7x7	1654	1,46	1,29	2,99	3,71
8x8	2210	15,51	5,72	65,47	25,68
9x9	2846	88,55	54,33	562,44	34,34
10x10	3562	429,08	363,40	1740,51	94,80
11x11	4358	6621,90	1565,46	3408,86	152,40
12x12	5234	>10000	>10000	>10000	212,91
13x13	6190	-	-	-	352,40
14x14	7226	-	-	-	616,90
15x15	8342	-	-	-	809,53
16x16	9538	-	-	-	1521,76
17x17	10814	-	-	-	1758,83
18x18	12170	-	-	-	2624,20
19x19	13606	-	-	-	3799,69
20x20	15122	-	-	-	5063,98
21x21	16718	-	-	-	5261,71
22x22	18394	-	-	-	7988,12
23x23	20150	-	-	-	>10000

(Tempos de execução medidos em segundos.)

5.5 Análise dos Resultados

De acordo com os resultados apresentados, quatro observações importantes podem ser feitas a respeito dos mesmos. Primeiro, o tempo de resolução não cresce monotonicamente em relação ao número de cláusulas, ou seja, o tempo utilizado para provar a equivalência entre dois multiplicadores *Wallace* com operandos de 20 bits é maior do que o tempo utilizado para provar a equivalência entre multiplicadores da mesma arquitetura com operandos de 21 bits. Portanto, analisar o desempenho de resolvedores por meio de apenas uma instância de uma classe de circuito pode resultar em conclusões incorretas. Essa é a principal razão pela qual a BenCGen gera circuitos de tamanhos parametrizados.

Segundo, não é possível prever o comportamento de um resolvedor de SAT utilizando instâncias de tamanho fixo, por exemplo, escolher circuitos com operandos de 16 bits ou 32 bits para avaliar desempenho de ferramentas. Se isso fosse feito, tamanhos específicos de circuitos poderiam ser selecionados de modo a favorecer um determinado resolvedor.

Terceiro, como cada resolvedor de SAT implementa suas próprias heurísticas, é

Tabela 5.5: Resultados para CEC de duas cópias de um mesmo multiplicador *Dadda Tree*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
3x3	518	0,00	0,00	0,01	0,00
4x4	956	0,01	0,01	0,01	0,00
5x5	1570	0,76	0,08	0,06	0,05
6x6	2376	16,40	0,73	0,75	0,37
7x7	3390	183,96	4,15	9,38	5,40
8x8	4628	1871,96	48,96	157,67	146,11
9x9	6106	8385,64	323,79	1981,53	196,09
10x10	7840	>10000	2933,220	>10000	658,38
11x11	9846	-	>10000	-	964,95
12x12	12140	-	-	-	1537,11
13x13	14738	-	-	-	1581,84
14x14	17656	-	-	-	2424,91
15x15	20910	-	-	-	5247,24
16x16	24516	-	-	-	8221,96
17x17	28490	-	-	-	>10000

(Tempos de execução medidos em segundos.)

difícil encontrar algum resolvedor que seja sempre o mais rápido em qualquer classe de circuito, e, muito menos, em qualquer tamanho de circuito dentre todas as classes. Em contra-partida, o que quase sempre ocorre é que um resolvedor de SAT apresenta resultados melhores na maior parte das instâncias de tamanhos diferentes para uma mesma classe de circuitos e que o mesmo é o mais rápido na maior parte das instâncias. Por isso, ao analisar o desempenho de uma ferramenta para verificação de equivalência, uma perspectiva global dos resultados deve ser observada.

Quarto, a classe de circuito a ser verificada é um fator muito mais importante para se determinar a dificuldade de uma instância do que o número de cláusulas. Com qualquer um dos resolvedores SAT é possível resolver instâncias de somadores muito mais rápido do que instâncias de multiplicadores, mesmo ambas possuindo o mesmo número de cláusulas.

Além dessas observações a respeito dos resultados, dois aspectos muito importantes em relação à utilização de resolvedores de SAT merecem atenção; principalmente levando-se em consideração a seguinte classificação de desempenho dos resolvedores de SAT para os circuitos gerados pela BenCGen: Berkmin561, Siege_v4, Minisat e, finalmente, Rsat.

O primeiro aspecto é que, embora a competição internacional de SAT ofereça informações úteis a respeito dos resolvedores de SAT atuais, essa competição não deve ser utilizada como única fonte de informação para aqueles que utilizam resolvedores

de SAT; principalmente para os que desenvolvem novas ferramentas e algoritmos para CEC. A partir dos resultados obtidos na Seção 5.4, fica claro que, para aqueles circuitos que são muito comuns em problemas de CEC, o Berkmin561 e o Siege_v4 são significativamente melhores do que o Minisat. Além disso, os resultados mostram que o Minisat ainda é ligeiramente melhor do que o Rsat. No entanto, o site da Competição SAT classifica o Rsat como o melhor resolvedor e o Minisat como o segundo melhor resolvedor. Certamente, o Berkmin561 e o Siege_v4 não estão listados porque os mesmos não competiram em 2007, mas provavelmente os primeiros lugares do Rsat e Minisat seriam mantidos devido às instâncias utilizadas.

O segundo aspecto, que é extremamente importante, é que em algumas publicações recentes sobre CEC como [MCBE06][WLLH07] em conferências importantes de EDA, o Minisat é utilizado como referência de desempenho para problemas de verificação de equivalência através de resolvedores de SAT. Entretanto, principalmente para problemas de CEC, não se deveria eleger um resolvedor como referência para CEC sem verificar se o mesmo é o mais adequado para os tipos de instâncias em que se está trabalhando. Além disso, se por alguma razão essa análise não puder ser feita, a referência de resolvedor de SAT para CEC deveria ser o Berkmin561 *s 1*, o que seria mais coerente com os resultados apresentados na Seção 5.4. Certamente, essa mudança de referência do Minisat para o Berkmin561 promoverá um comparação mais justa entre as novas técnicas do estado da arte para CEC e as técnicas que utilizam apenas o resolvedor de SAT.

Tabela 5.6: Resultados para CEC de duas cópias de um mesmo divisor *Restoring*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
6 / 3	975	0,01	0,01	0,01	0,00
8 / 4	1680	0,61	0,05	0,62	0,02
10 / 5	2577	0,42	0,23	0,94	0,17
12 / 6	3666	2,43	2,74	6,04	0,91
14 / 7	4947	28,57	33,66	34,96	4,43
16 / 8	6420	153,17	382,55	116,17	15,32
18 / 9	8085	1565,53	2372,39	194,66	16,58
20 / 10	9942	>5000	>5000	260,26	24,56
22 / 11	11991	-	-	311,99	65,87
24 / 12	14232	-	-	392,14	127,89
26 / 13	16665	-	-	474,49	231,75
28 / 14	19290	-	-	550,86	404,26
30 / 15	22107	-	-	626,14	347,26
32 / 16	25116	-	-	703,62	645,32
34 / 17	28317	-	-	787,33	780,13
36 / 18	31710	-	-	914,27	1047,45
38 / 19	35295	-	-	962,33	1809,22
40 / 20	39072	-	-	1071,31	2255,54
42 / 21	43041	-	-	1226,55	2604,58
44 / 22	47202	-	-	1318,74	3617,73
46 / 23	51555	-	-	1445,89	>5000
48 / 24	56100	-	-	1529,41	-
50 / 25	60837	-	-	1665,51	-
52 / 26	65766	-	-	1838,74	-
54 / 27	70887	-	-	1964,66	-
56 / 28	76200	-	-	2067,98	-
58 / 29	81705	-	-	2246,90	-
60 / 30	87402	-	-	2359,22	-
62 / 31	93291	-	-	2514,50	-
64 / 32	99372	-	-	2676,37	-
...	...	-	-	...	-

(Tempos de execução medidos em segundos.)

Tabela 5.7: Resultados para CEC de duas cópias de um mesmo divisor *Non Restoring*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
6 / 3	442	0,01	0,06	0,01	0,00
8 / 4	755	0,05	0,04	0,04	0,01
10 / 5	1152	0,30	0,34	0,47	0,22
12 / 6	1633	3,76	2,66	4,66	4,24
14 / 7	2198	103,16	19,90	30,63	25,03
16 / 8	2847	408,92	327,86	581,20	30,67
18 / 9	3580	3600,00	2519,03	3734,54	79,77
20 / 10	4397	>10000	>10000	>10000	97,46
22 / 11	5298	-	-	-	148,69
24 / 12	6283	-	-	-	196,06
26 / 13	7352	-	-	-	212,96
28 / 14	8505	-	-	-	582,03
30 / 15	9742	-	-	-	737,91
32 / 16	11063	-	-	-	1118,49
34 / 17	12468	-	-	-	1454,70
36 / 18	13957	-	-	-	2048,92
38 / 19	15530	-	-	-	2364,63
40 / 20	17187	-	-	-	3392,10
42 / 21	18928	-	-	-	5138,07
44 / 22	20753	-	-	-	5802,30
46 / 23	22662	-	-	-	7988,44
48 / 24	24655	-	-	-	>10000

(Tempos de execução medidos em segundos.)

Tabela 5.8: Resultados para CEC de duas cópias de um mesmo somador *Ripple Carry*

Numero de bits	Numero de Clásulas	Rsat	Minisat	Siege_v4	Berkmin561
128	5271	0,13	0,16	0,15	0,28
256	10519	0,69	0,65	0,78	1,39
384	15767	1,99	1,83	2,15	5,79
512	21015	4,20	2,99	3,75	11,11
640	26263	7,39	3,78	6,62	16,90
768	31511	10,86	5,73	10,14	23,69
896	36759	13,57	9,51	13,01	32,38
1024	42007	24,94	11,96	17,58	49,23
1152	47255	24,34	17,22	22,43	68,74
1280	52503	38,93	21,67	28,86	91,29
1408	57751	44,59	21,89	33,46	109,94
1536	62999	62,13	28,04	41,03	125,88
1664	68247	75,58	37,92	47,82	173,34
1792	73495	85,24	36,25	56,82	235,09
1920	78743	90,39	44,76	65,65	272,47
2048	83991	127,74	58,88	79,64	312,35
...	...	...	...	...	...

(Tempos de execução medidos em segundos.)

Capítulo 6

Resultados

6.1 Introdução

Este capítulo apresenta os resultados obtidos aplicando a técnica de pré-processamento que deriva informações redundantes apresentada no Capítulo 4 e compara-os com os resultados obtidos por meio das principais técnicas do estado da arte de Verificação de Equivalência de Circuitos Combinacionais (CEC, do Inglês, *Combinational Equivalence Checking*) baseado em Satisfabilidade (SAT). Como todas as técnicas de pré-processamento necessitam de um resolvedor de SAT como suporte, será utilizado o resolvedor que obteve o melhor desempenho em CEC, o Berkmin561, como mostrado no Capítulo 5.

6.2 Características dos Experimentos

Os resultados apresentados foram obtidos utilizando computador com um processador um Athlon IV 3,0 GHz com 2 GB de memória e com o sistema operacional Linux distribuição Fedora Core 10.0. Todas as tabelas seguintes apresentam os tempos de resolução em segundos para as ferramentas selecionadas em problemas de CEC entre um circuito original e a sua cópia sintetizada. A síntese dos circuitos apresentados foi feita utilizando a ferramenta ABC [Ber05]. Como ferramentas de pré-processamento são utilizadas, os tempos consumidos pelas mesmas são sempre somados ao tempo usado pelo resolvedor Berkmin561.

6.3 Seleção de Ferramentas do Estado da Arte de CEC Baseado em SAT

Para estabelecer um comparativo com a ferramenta proposta, a qual é representada pelo conjunto Vimplic + Berkmin561, as seguintes ferramentas do estado da arte de

CEC baseado em SAT foram selecionadas:

- o resolvedor Berkmin561 sem a utilização de técnica de pré-processamento;
- o C-SAT [LWC+03];
- o NiVER [SP04] + Berkmin561; e
- o SatElite [EB05] + Berkmin561.

O número de ferramentas escolhidas para a comparação foi limitado em 4 devido ao grande número de instâncias utilizadas no benchmark; o que resultou em um tempo total de simulação contínua de aproximadamente 100 dias, excluindo-se o tempo de preparação das instâncias. Dentre as ferramentas do estado da arte de CEC baseado em SAT, a lista acima foi selecionada pelos seguintes motivos. O Berkmin561 é o melhor resolvedor de SAT conhecido para instâncias de CEC; o C-SAT é o resolvedor de SAT para circuitos mais eficiente disponibilizado até o momento; o NiVER foi escolhido por ser capaz de superar o desempenho ou no mínimo oferecer desempenho similar ao da ferramenta Hypre [BW03] na maior parte das instâncias apresentadas pelos autores da NiVER [SP04]; e, finalmente, o SatELite é a mais recente ferramenta de pré-processamento disponibilizada. O resolvedor Berkmin561 foi executado com o parâmetro *s 1* que ativa uma estratégia especial para verificação de equivalência e o C-SAT foi executado com a opção *-u*.

Outras ferramentas também foram solicitadas como a HANNIBAL [Kun93], a ferramenta publicada por Burch [BS98], a TIP [TGA00], a ferramenta publicada por Goldberg [GPB01], a IMP2C [AH04], a ferramenta publicada por Stoffel [SK04] e a QuteSAT [WLLH07]. Infelizmente, as duas primeiras ferramentas, por terem sido elaboradas a mais de 10 anos, não se encontram mais disponíveis segundo os próprios autores. Para a ferramenta TIP, nem o autor e nem o instituto onde ele a desenvolveu possuem alguma versão da mesma. A ferramenta publicada por Goldberg e a publicada por Stoffel, apesar de serem mais recentes, também não se encontram disponíveis segundo os autores. A ferramenta IMP2C nunca foi disponibilizada, nem por solicitação direta aos autores, mesmo tendo obtido resultados muitos bons segundo os experimentos deles. Segundo os autores da ferramenta QuteSAT, o código está sendo reestruturado atualmente e, por isso, não está disponível, nem mesmo a versão relativa à publicação de 2007 tornou-se pública.

6.4 Resultados Utilizando o Benchmark ISCAS 85

O primeiro benchmark selecionado foi o ISCAS 85, já que esse é o benchmark mais popular na área de CEC e também porque o mesmo fornece apenas circuitos combinacionais, que é o foco do presente trabalho. A Tabela 6.1 mostra os resultados

Tabela 6.1: Resultados para Instâncias de CEC derivadas do Benchmark ISCAS 85

Circuito	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK
c1355	**0,10**	0,76	0,43	0,14	0,18
c1908	**0,08**	0,11	0,10	0,93	0,25
c2670	0,29	0,20	**0,15**	0,93	0,36
c3540	0,89	5,35	**0,81**	0,85	2,39
c432	**0,01**	0,01	0,02	0,04	0,04
c499	**0,08**	0,38	0,09	0,13	0,14
c5315	1,37	2,04	**1,01**	1,35	1,24
c6288	>1000	>1000	>1000	>1000	>1000
c7552	3,34	3,21	**2,05**	2,06	3,55
c880	**0,03**	0,11	0,04	0,12	0,08

(Tempos de execução medidos em segundos.)

obtidos utilizando o Berkmin561 (BK), o C-SAT, o conjunto NiVER + Berkmin561 (NVR+BK), o conjunto SatElite + Berkmin561 (STL+BK) e o conjunto Vimplic + Berkmin561 (VMPL+BK) em problemas de CEC entre um circuito e a sua versão sintetizada. O melhor tempo de resolução está destacado em negrito e o tempo total foi limitado em 1000 segundos.

A Tabela 6.1 mostra que a melhor ferramenta para as instâncias de CEC derivadas do ISCAS 85 foi o Berkmin561 sem pré-processamento. Essa ferramenta é capaz de resolver 7 das 10 instâncias em décimos segundos. Além disso, o Berkmin561 supera todas as outras ferramentas em 5 das 9 instâncias resolvidas, seguido pelo NiVER + Berkmin561 que possui o melhor tempo em 4 das 9 instâncias resolvidas. É interessante notar que, mesmo quando o Berkmin561 não apresenta o menor tempo, o mesmo é apenas frações de segundo mais lento do que o NiVER + Berkmin561. Infelizmente, o C-SAT, o SatElite + Berkmin561 e o Vimplic + Berkmin561 não ofereceram bons resultados.

Após analisar esses resultados, duas observações podem ser feitas. A primeira e mais importante é que ferramentas de pré-processamento não devem ser utilizadas para instâncias de circuitos muito pequenos, já que, nesses casos, o resolvedor de SAT sozinho é suficiente para oferecer resultados muito bons como os apresentados na Tabela 6.1. Além disso, o pré-processamento pode, na realidade, aumentar o tempo total de resolução como aconteceu com as ferramentas SatElite e Vimplic nessas instâncias.

A segunda observação é que tanto a ferramenta SatElite quanto a Vimplic consomem mais tempo no pré-processamento com o objetivo de gerar uma fórmula mais simples para o resolvedor de SAT. Em contrapartida, o tempo de pré-processamento utilizado pela ferramenta NiVER é praticamente desprezível quando comparado com o tempo de resolução das instâncias com o resolvedor de SAT e, por isso, a fórmula gerada por essa ferramenta provavelmente não deveria ser tão fácil para o resolvedor quanto as fórmulas produzidas pela SatElite e a Vimplic. Essa é a razão pela qual apenas a ferramenta NiVER consegue melhorar o tempo total de resolução das instâncias de CEC criadas a partir dos circuitos do ISCAS 85, pois a NiVER praticamente não

gasta tempo na etapa de pré-processamento.

Como detalhado na Seção 5.2, embora o ISCAS 85 tenha sido utilizado por mais de duas décadas e, devido à sua popularidade, tenha sido selecionado para avaliar as ferramentas de CEC nesse capítulo, esse benchmark está claramente desatualizado. Portanto, outro benchmark foi necessário.

Outros importantes benchmarks como o benchmark ISCAS 89 [FBK89], o benchmark do Politecnico di Torino [CRS00] e o benchmark do IWLS 2005 [Alb05] não são muito adequados para CEC já que quase a totalidade dos circuitos apresentados nesses benchmarks é de circuitos seqüenciais e, por isso, Verificação de Equivalência de Circuitos Seqüenciais e não Verificação de Equivalência de Circuitos Combinacionais deve ser utilizada para se obter os menores tempos de resolução[1]. Além disso, o último benchmark listado que é o mais completo disponibiliza a maior parte dos circuitos em RTL, o que é inadequado, pois muitas ferramentas importantes de CEC, tal como a C-SAT, exigem que o formato de entrada seja em nível de portas lógicas (BENCH).

De modo a prover um comparativo e uma análise experimental mais adequada e completa das ferramentas do estado da arte de CEC baseado em SAT, o presente trabalho utilizou a ferramenta BenCGen descrita no Capítulo 5.

6.5 Resultados Utilizando um Benchmark Produzido pela BenCGen

Por meio da ferramenta BenCGen, um benchmark composto por diversos circuitos combinacionais de classes e tamanhos variados foi gerado e os resultados de tempo de cada uma das ferramentas analisadas é disponibilizado a seguir. O tempo limite para a resolução de instâncias de multiplicadores foi limitado em 30.000 segundos e para as instâncias divisores, 10.000 segundos. O tamanho dos circuitos multiplicadores foi limitado em 32 bits para cada operando e para os somadores em 1024 bits. Instâncias muito pequenas (com tempos de resolução de menos de um segundo) foram removidas das tabelas porque, nesses casos, como descrito anteriormente, nenhum pré-processamento é necessário.

Duas importantes observações descritas no Capítulo 5 devem ser ressaltadas antes da apresentação dos resultados. A primeira é que o tempo de resolução não cresce monotonicamente de acordo com o tamanho do circuito já que o resolvedor de SAT é um procedimento de busca com heurísticas complexas. A segunda é que nenhuma

[1]É importante notar que circuitos seqüenciais podem ser provados corretos com Verificação de Equivalência de Circuitos Combinacionais. Para fazê-lo, basta cortar o circuito em torno dos *latches* ou *flip-flops* e transformar, em entradas primárias e saídas primárias do circuito, as saídas e entradas dos *latches* ou *flip-flops* respectivamente. No entanto, como destacado anteriormente, o presente trabalho concentra-se apenas em abordagens de CEC para circuitos combinacionais.

das cinco ferramentas analisadas nesse capítulo é capaz de superar todas as outras no benchmark completo. A classificação de desempenho varia de acordo com o tamanho e o tipo da instância. Por este motivo, para se eleger a ferramenta mais eficiente, uma perspectiva global dos resultados deve ser analisada em detrimento de se concentrar em uma classe ou tamanho específico de circuito.

As Tabelas 6.2 a 6.6 apresentam os resultados de CEC de somadores *Carry Save*, *Carry Skip*, *Carry Select*, *Block Carry Lookahead* e *Ripple Carry* usando as ferramentas selecionadas. A Tabela 6.2 mostra os resultados para os somadores *Carry Save*. Para o somador de 128 bits, o resultado do Vimplic + Berkmin561 é 12% mais lento do que a utilização do Berkmin561 sem pré-processamento. Entretanto, nas 14 instâncias seguintes, o Vimplic + Berkmin561 é capaz de resolvê-las de 27,9% a 152,2% mais rápido do que o Berkmin561 sozinho. Superar o Berkmin561 sozinho é um grande resultado porque esse resolvedor já é capaz de oferecer melhor desempenho do que importantes resolvedores como o Minisat, o RSat e o Siege_v4 em instâncias de CEC como mostrado no Capítulo 5. Já em relação ao C-SAT, o tempo de resolução não pôde ser comparado nessas instâncias, pois essa ferramenta sempre imprime um tempo de resolução de 0,0 segundos, mesmo utilizando um parâmetro na linha de comando para ignorar a estrutura topológica do circuito. Apesar de ser um comportamento inesperado para essa ferramenta, tal fato já foi relatado por outros autores [WLLH07].

Continuando na Tabela 6.2, o Vimplic + Berkmin561 é mais lento do que o NiVER + Berkmin561 nas duas menores instâncias em 33,8% e 17,2% respectivamente; porém o mesmo é mais rápido de 14,6% a 258,1% nas outras 13 instâncias restantes. Finalmente, o Vimplic + Berkmin561 é de 8,4% a 65,7% mais eficiente do que o SatElite + Berkmin561 em todas as 15 instâncias mostradas. Assim, pode-se afirmar que, dentre as cinco ferramentas apresentadas, a Vimplic certamente oferece os melhores resultados. Note que, na instância do somador de 448 bits, a ferramenta NiVER apresenta uma falha em sua execução.

Os resultados dos somadores *Carry Skip* apresentados na Tabela 6.3 são bem similares aos resultados apresentados na Tabela 6.2, pois o Vimplic + Berkmin561 é capaz de superar todas as outras ferramentas em todas as instâncias com apenas algumas exceções. É importante notar que os resultados são ainda melhores do que os obtidos com os somadores *Carry Save*. O Vimplic + Berkmin561 supera o Berkmin561 em todas as instâncias apresentando um desempenho de 15,5% a 52,1% melhor do que o Berkmin561 sozinho; supera também o C-SAT em todas as instâncias com um desempenho de 0,6% a 380,5% melhor; e ainda supera novamente, em todas as instâncias, o NiVER + Berkmin61 com uma melhoria de 21,2% a 359,6% no tempo total de execução. Em relação ao SatElite + Berkmin561, dentre as 15 instâncias, o Vimplic + Berkmin561 é melhor em 13, perdendo em uma e ficando em igualdade em outra; proporcionando uma melhoria no desempenho de 3,1% a 33,9%.

Tabela 6.2: Resultados para Instâncias de CEC de Somadores *Carry Save* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
128	1,3	0,0	0,9	1,5	1,4	-12,0	0,0	-33,8	8,4
192	3,5	0,0	1,7	3,0	2,0	73,9	0,0	-17,2	49,9
256	4,0	0,0	3,6	5,2	3,2	27,9	0,0	14,6	65,7
320	8,5	0,0	7,1	8,0	5,9	44,9	0,0	19,9	36,1
384	13,8	0,0	17,1	11,7	8,6	60,2	0,0	98,5	35,7
448	21,7	0,0	FALHA	14,6	11,7	85,0	0,0	FALHA	24,4
512	26,5	0,0	36,7	18,9	15,6	69,8	0,0	134,5	20,7
576	40,5	0,0	54,9	23,3	19,7	105,8	0,0	178,8	18,4
640	47,9	0,0	88,2	29,9	24,6	94,5	0,0	258,1	21,5
704	64,8	0,0	96,7	37,6	30,7	111,3	0,0	215,0	22,6
768	79,4	0,0	131,4	42,4	37,3	112,5	0,0	251,9	13,5
832	104,4	0,0	152,6	52,3	44,7	133,8	0,0	241,8	17,1
896	123,5	0,0	106,5	61,0	52,7	134,1	0,0	101,9	15,6
960	157,0	0,0	129,0	72,2	62,3	152,2	0,0	107,2	16,0
1024	172,8	0,0	143,6	85,2	74,9	130,8	0,0	91,8	13,8

(Tempos de execução medidos em segundos.)

Tabela 6.3: Resultados para Instâncias de CEC de Somadores *Carry Skip* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
128	1,0	0,9	1,5	0,8	0,9	16,9	0,6	65,2	-15,7
192	2,3	2,3	2,1	1,7	1,7	32,9	32,7	21,2	-0,1
256	3,0	3,9	4,6	2,7	2,4	28,3	66,5	94,1	14,6
320	4,5	6,8	8,2	4,8	3,6	26,5	90,7	128,1	33,9
384	7,0	12,5	13,2	7,0	5,8	21,8	115,6	127,9	20,9
448	11,0	19,4	18,3	10,0	8,9	23,4	118,5	105,4	12,0
512	14,3	30,8	25,5	13,1	10,9	30,6	181,7	133,3	19,5
576	20,1	38,5	36,1	18,2	15,0	34,0	157,2	140,7	21,5
640	23,9	51,2	46,9	22,3	18,4	29,5	177,9	154,4	21,3
704	30,7	47,9	68,1	26,0	20,2	52,1	137,4	237,9	29,0
768	36,7	103,4	107,2	32,7	31,8	15,5	225,5	237,6	3,1
832	46,0	120,3	117,8	38,4	33,1	38,8	263,1	255,7	16,0
896	62,4	160,3	149,5	49,6	46,3	34,9	246,6	223,1	7,2
960	72,8	196,4	141,3	58,6	52,4	38,9	274,7	169,7	11,9
1024	72,4	239,1	228,7	65,7	49,8	45,4	380,5	359,6	32,0

(Tempos de execução medidos em segundos.)

Tabela 6.4: Resultados para Instâncias de CEC de Somadores *Carry Select* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
128	1,8	1,5	2,4	1,4	2,1	-14,5	-31,3	13,6	-35,3
192	4,0	3,3	5,1	3,2	3,5	15,9	-4,0	47,3	-7,2
256	6,3	7,6	9,6	5,9	6,7	-5,4	13,1	43,3	-12,5
320	10,6	13,0	18,5	9,2	10,3	3,7	27,0	80,0	-10,2
384	17,5	22,7	30,7	15,0	15,3	14,5	48,5	100,7	-2,0
448	22,9	33,8	44,1	17,6	22,9	0,0	47,4	92,5	-23,0
512	31,6	50,7	63,7	25,7	29,2	8,2	74,0	118,4	-11,9
576	43,7	76,5	88,4	32,0	37,6	16,2	103,5	135,0	-14,8
640	51,8	98,5	128,3	42,2	49,0	5,9	101,1	161,9	-13,8
704	76,7	140,6	160,8	48,9	63,9	19,9	120,0	151,5	-23,5
768	79,2	182,8	215,4	62,4	70,4	12,5	159,7	206,1	-11,3
832	96,0	222,7	281,6	72,3	84,7	13,4	163,0	232,7	-14,6
896	117,1	308,2	304,0	100,6	96,3	21,6	220,1	215,8	4,5
960	137,3	405,5	414,8	103,4	120,5	14,0	236,6	244,4	-14,2
1024	153,1	440,3	448,9	119,6	133,6	14,6	229,7	236,1	-10,5

(Tempos de execução medidos em segundos.)

Os resultados dos somadores *Carry Select* são apresentados na Tabela 6.4. Nessas instâncias, o Vimplic + Berkmin561 supera o Berkmin561 sozinho em 12 das 15 instâncias oferecendo um ganho de desempenho de 3,7% a 21,6%; supera também o C-SAT em 13 das 15 instâncias com uma vantagem variando de 13,1% a 236,6%; e ainda supera o NiVER + Berkmin561 em todas as 15 instâncias resolvendo-as de 13,6% a 244,4% mais rápido. Contudo, o Vimplic + Berkmin561 é em média 13,1% mais lento do que o SatElite + Berkmin561 nessas instâncias. É interessante notar que o SatElite + Berkmin561 é apenas um pouco melhor do que o Vimplic + Berkmin561 se for levado em consideração as diferenças de tempo entre o Vimplic + Berkmin561 e o SatElite + Berkmin561 nas outras instâncias em que os resultados da ferramenta proposta foram melhores.

Os resultados dos somadores *Block Carry LookAhead* são apresentados na Tabela 6.5. Assim como nos resultados apresentados nas Tabelas 6.2 e 6.3, o Vimplic + Berkmin561 é capaz de superar todas as outras ferramentas em quase todas as instâncias. O ganho de desempenho do Vimplic + Berkmin561 em relação ao Bermim561, ao C-SAT e ao NiVER + Berkmin561 variam de 0,6% a 51,4%, 17,0% a 298,9% e 56,4% a 149,5%, respectivamente. Em relação ao SatElite + Berkmin561, o Vimplic + Berkmin561 é melhor, no entanto, por uma margem bem pequena.

A Tabela 6.6 mostra os resultados para os somadores *Ripple Carry*. Nessas instâncias, o Vimplic + Berkmin561 supera o Berkmin561 sozinho em 13 das 15 instâncias oferecendo um ganho de desempenho de 0,4% a 50,7%; supera também o NiVER + Berkmin561 em todas as 15 instâncias com uma vantagem variando de 26,5% a 339,3%; e ainda supera o SatElite + Berkmin561 resolvendo 13 das 15 instâncias de 1,8% a 46,3% mais rápido. Contudo, o Vimplic + Berkmin561 é em média 31,7% mais lento do que

Tabela 6.5: Resultados para Instâncias de CEC de Somadores *Block Carry LookAhead* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
128	1,1	1,0	1,5	0,9	0,7	51,4	35,1	95,9	23,9
192	2,0	2,0	2,8	1,8	1,7	14,0	17,0	61,4	8,0
256	3,6	5,0	4,7	3,5	2,9	24,2	70,6	61,1	20,1
320	5,1	8,0	8,5	5,1	5,2	-2,9	53,6	62,4	-1,5
384	8,8	14,0	12,6	8,2	8,0	10,4	75,2	57,2	2,3
448	11,3	21,0	17,5	11,4	11,2	0,6	87,7	56,4	2,1
512	15,2	33,0	26,2	14,2	14,7	3,5	124,9	78,7	-3,0
576	20,6	46,0	36,7	18,3	18,0	14,2	155,6	103,9	1,8
640	28,0	61,0	48,8	23,0	22,5	24,3	171,0	116,7	2,2
704	35,6	86,0	61,1	31,3	32,8	8,6	162,2	86,3	-4,7
768	42,0	111,0	73,7	37,1	37,0	13,7	200,3	99,3	0,4
832	52,6	133,0	89,0	42,0	42,2	24,7	215,5	111,1	-0,3
896	57,9	188,0	118,0	50,2	47,3	22,5	297,4	149,5	6,0
960	78,7	224,0	148,7	59,2	65,5	20,2	242,0	127,0	-9,6
1024	93,0	284,8	155,3	65,6	71,4	30,3	298,9	117,5	-8,2

(Tempos de execução medidos em segundos.)

o C-SAT nessas instâncias.

As Tabelas 6.7 a 6.11 apresentam os resultados de CEC de multiplicadores *Wallace Tree*, *Reduced Tree*, *Carry LookAhead*, *Array* e *Dadda Tree* usando as ferramentas selecionadas. Os multiplicadores foram escolhidos pois os mesmos podem ser considerados a classe mais difícil para a verificação dentre todos os circuitos combinacionais.

A Tabela 6.7 mostra os resultados para instâncias de CEC de multiplicadores *Wallace Tree*. Além de superar o desempenho de todas as ferramentas em praticamente todas as instâncias apresentadas, o Vimplic + Berkmin561 é o único que consegue resolver até a instância de multiplicadores com operandos de 32 bits no tempo limite de 30.000 segundos. O Berkmin561 resolve apenas até multiplicadores com operandos de 23 bits, o C-SAT apenas até 12 bits, o NiVER + Berkmin561 apenas até 30 bits e, finalmente, o SatElite apenas até 28 bits. O ganho de desempenho do Vimplic + Berkmin561 varia de 20,5% a 637,1% em relação ao Berkmin561, varia de 859,0% a 4580,2% em relação ao C-SAT, varia de 4,2% a 253,2% em relação ao NiVER + Berkmin561 e varia de 1,1% a 378,3% em relação ao SatElite + Berkmin561. Esses resultados demonstram a grande vantagem oferecida pelo Vimplic + Berkmin561 em relação às ferramentas do estado da arte de CEC baseado em SAT.

A Tabela 6.8 mostra os resultados para instâncias de CEC de multiplicadores *Reduced Tree*. Novamente, o Vimplic + Berkmin561 é o único que consegue resolver as maiores instâncias e, além disso, supera todas as outras ferramentas em todas as instâncias apresentadas, com exceção da instância com operandos de 18 bits em que o Vimplic + Berkmin561 fica apenas atrás do NiVER + Berkmin561. Nessas instâncias, o aumento de desempenho médio ainda foi maior do que os apresentados na Tabela 6.7. O ganho de desempenho do Vimplic + Berkmin561 em relação ao Bermim561, ao C-SAT, ao NiVER + Berkmin561 e ao SatElite + Berkmin561 variam de 100,1% a

Tabela 6.6: Resultados para Instâncias de CEC de Somadores *Ripple Carry* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
128	0,8	0,2	1,1	1,0	0,8	-1,2	-74,0	26,5	19,6
192	1,3	0,5	1,8	1,4	1,4	-9,3	-63,9	30,0	-0,9
256	2,4	1,0	4,0	2,4	2,4	0,4	-56,5	68,5	1,8
320	3,5	1,9	7,2	3,6	3,3	3,6	-42,9	114,7	7,5
384	6,7	3,1	10,9	6,4	4,4	50,7	-30,5	147,5	45,0
448	8,2	4,5	19,5	9,3	7,2	14,4	-37,2	170,6	28,5
512	11,0	6,5	24,2	11,7	9,7	14,0	-32,5	150,1	20,4
576	13,2	9,1	34,7	15,3	11,3	16,2	-19,6	205,6	35,3
640	17,3	12,3	54,9	22,2	18,3	-5,7	-32,7	199,5	21,3
704	24,5	16,4	59,6	28,2	19,3	27,0	-15,0	209,4	46,3
768	30,3	20,8	66,8	31,8	23,4	29,4	-11,0	185,7	36,1
832	36,5	25,2	129,2	38,9	29,4	24,0	-14,4	339,3	32,2
896	46,1	31,3	151,9	43,0	39,6	16,4	-21,0	283,4	8,5
960	48,5	39,4	161,1	41,4	47,9	1,3	-17,8	236,6	-13,6
1024	55,4	48,9	191,5	62,8	52,2	6,1	-6,3	266,7	20,3

(Tempos de execução medidos em segundos.)

Tabela 6.7: Resultados para Instâncias de CEC de Multiplicadores *Wallace Tree* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
8x8	75,6	37,0	73,2	9,4	62,7	20,5	-41,0	16,7	-85,0
9x9	853,2	258,0	374,0	275,3	272,3	213,4	-5,2	37,4	1,1
10x10	926,5	1798,0	326,9	367,1	187,5	394,2	859,0	74,4	95,8
11x11	1023,0	7200,0	543,3	506,5	153,8	565,0	4580,2	253,2	229,3
12x12	1295,9	11573,4	615,1	1094,7	286,1	353,0	3945,7	115,0	282,7
13x13	2657,1	>30000	1066,7	1925,6	429,1	519,3	-	148,6	348,8
14x14	4279,0	-	1242,8	3174,5	663,8	544,7	-	87,2	378,3
15x15	7672,3	-	1442,7	4409,0	1040,9	637,1	-	38,6	323,6
16x16	6482,3	-	2369,2	4490,0	1130,0	473,7	-	109,7	297,4
17x17	6173,4	-	2924,7	5463,0	1257,0	391,1	-	132,7	334,6
18x18	6342,3	-	3268,9	4863,5	1424,5	345,2	-	129,5	241,4
19x19	8404,6	-	3886,4	6618,8	2366,9	255,1	-	64,2	179,6
20x20	11758,4	-	6348,2	10198,2	2818,0	317,3	-	125,3	261,9
21x21	11240,9	-	6117,5	8832,1	3387,3	231,9	-	80,6	160,7
22x22	15704,5	-	8055,5	13327,9	4386,6	258,0	-	83,6	203,8
23x23	19324,8	-	10532,0	15106,4	5916,8	226,6	-	78,0	155,3
24x24	>30000	-	11406,7	19727,0	7109,7	-	-	60,4	177,5
25x25	-	-	13120,6	19028,9	8870,8	-	-	47,9	114,5
26x26	-	-	11790,3	18536,0	8430,5	-	-	39,9	119,9
27x27	-	-	12851,8	20357,0	10178,8	-	-	26,3	100,0
28x28	-	-	15213,2	22408,0	14602,3	-	-	4,2	53,5
29x29	-	-	25841,3	>30000	15666,7	-	-	64,9	-
30x30	-	-	22987,5	-	16040,1	-	-	43,3	-
31x31	-	-	>30000	-	20315,0	-	-	-	-
32x32	-	-	-	-	27667,9	-	-	-	-

(Tempos de execução medidos em segundos.)

Tabela 6.8: Resultados para Instâncias de CEC de Multiplicadores *Reduced Tree* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
8x8	70,8	39,1	53,0	68,7	35,4	100,1	10,4	49,7	94,2
9x9	321,9	305,6	136,6	220,0	83,3	286,3	266,8	63,9	164,0
10x10	550,6	1848,2	177,5	134,6	93,1	491,4	1885,4	90,7	44,6
11x11	894,2	13783,6	152,8	336,6	100,8	786,9	13571,5	51,6	233,9
12x12	1940,3	>30000	291,8	573,5	258,6	650,3	-	12,8	121,8
13x13	2344,4	-	1123,7	1657,1	537,1	336,5	-	109,2	208,6
14x14	3693,4	-	1151,2	2286,2	901,3	309,8	-	27,7	153,7
15x15	9793,4	-	2818,3	3366,0	947,1	934,0	-	197,6	255,4
16x16	8245,6	-	2464,7	4523,7	1917,8	330,0	-	28,5	135,9
17x17	8218,3	-	3600,1	6042,6	2078,1	295,5	-	73,2	190,8
18x18	15952,3	-	4100,8	8004,6	4870,8	227,5	-	-15,8	64,3
19x19	19376,7	-	4707,8	9399,1	2812,7	588,9	-	67,4	234,2
20x20	16741,7	-	6473,7	8052,1	3061,2	446,9	-	111,5	163,0
21x21	17807,3	-	8533,0	10133,2	5213,1	241,6	-	63,7	94,4
22x22	>30000	-	10703,1	13657,3	8147,0	-	-	31,4	67,6
23x23	-	-	10322,9	13352,4	4981,6	-	-	107,2	168,0
24x24	-	-	13860,9	18903,1	10476,1	-	-	32,3	80,4
25x25	-	-	16118,5	24855,4	10596,4	-	-	52,1	134,6
26x26	-	-	23873,6	27721,3	10206,3	-	-	133,9	171,6
27x27	-	-	27546,5	>30000	10665,7	-	-	158,3	-
28x28	-	-	>30000	-	9535,2	-	-	-	-
29x29	-	-	-	-	11655,4	-	-	-	-
30x30	-	-	-	-	17441,7	-	-	-	-
31x31	-	-	-	-	20969,8	-	-	-	-
32x32	-	-	-	-	22900,5	-	-	-	-

(Tempos de execução medidos em segundos.)

934,0%, 10,4% a 13571,5%, 28,5% a 197,6% e 64,3% a 255,4%, respectivamente.

Os resultados para instâncias de CEC de multiplicadores *Carry LookAhead* são mostrados na Tabela 6.9. Nessas instâncias, o Vimplic + Berkmin561 supera o Berkmin561 sozinho todas as 15 instâncias oferecendo um ganho de desempenho de 22,2% a 121,4%; supera também o C-SAT com uma vantagem variando de 39,0% a 5112,6%; e ainda supera o SatElite + Berkmin561 em 13 das 15 instâncias resolvendo-as de 1,2% a 89,2% mais rápido. Em relação ao NiVER + Berkmin561, o Vimplic + Berkmin561 fica praticamente em igualdade, pois o Vimplic + Berkmin561 é mais rápido em 7 instâncias e mais lento em 8, mas, analisando-se o ganho de desempenho médio, a ferramenta proposta fica com uma vantagem média de 9,8%. É interessante notar que essas são as instâncias mais difíceis para os resolvedores já que a maior instância resolvida foi a relativa a multiplicadores com operandos de 23 bits.

Os resultados para instâncias de CEC de multiplicadores *Array* são mostrados na Tabela 6.10. A primeira instância mostra um ótimo desempenho do C-SAT o qual é de 575% a 798% mais rápido do que as outras ferramentas. No entanto, esse desempenho não se mantém para as outras instâncias e, mais uma vez, o Vimplic + Berkmin561 supera todas as outras ferramentas. O ganho de desempenho do Vimplic + Berkmin561 varia de 22,4% a 90,5% em relação ao Berkmin561, varia de 14,1% a 4733,6% em relação ao C-SAT, varia de 3,7% a 80,6% em relação ao NiVER + Berkmin561 e varia

Tabela 6.9: Resultados para Instâncias de CEC de Multiplicadores *Carry LookAhead* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
8x8	47,9	53,8	49,1	55,0	29,1	64,6	85,0	68,9	89,2
9x9	123,0	286,4	115,0	173,0	100,7	22,2	184,5	14,2	71,9
10x10	272,9	2481,6	214,3	192,0	137,5	98,5	1704,4	55,8	39,6
11x11	545,0	12830,9	376,3	369,8	246,2	121,4	5112,6	52,9	50,2
12x12	729,9	>30000	452,5	522,2	515,9	41,5	-	-12,3	1,2
13x13	1317,9	-	801,6	998,8	843,9	56,2	-	-5,0	18,4
14x14	2342,6	-	1456,9	1582,9	1196,3	95,8	-	21,8	32,3
15x15	3166,4	-	2303,4	2083,8	2361,3	34,1	-	-2,5	-11,8
16x16	4495,5	-	2850,6	3341,4	2705,0	66,2	-	5,4	23,5
17x17	6320,9	-	4340,7	4782,9	4513,6	40,0	-	-3,8	6,0
18x18	9492,7	-	5085,1	5981,0	5742,2	65,3	-	-11,4	4,2
19x19	12326,3	-	7318,8	9903,0	9083,0	35,7	-	-19,4	9,0
20x20	19008,5	-	9990,0	11746,5	11391,0	66,9	-	-12,3	3,1
21x21	23721,1	-	14346,9	15077,9	16779,8	41,4	-	-14,5	-10,1
22x22	>30000	-	19912,6	22075,8	18280,6	-	-	8,9	20,8
23x23	-	-	23674,1	>30000	>30000	-	-	-	-
24x24	-	-	>30000	-	-	-	-	-	-
25x25	-	-	-	-	-	-	-	-	-
26x26	-	-	-	-	-	-	-	-	-
27x27	-	-	-	-	-	-	-	-	-
28x28	-	-	-	-	-	-	-	-	-
29x29	-	-	-	-	-	-	-	-	-
30x30	-	-	-	-	-	-	-	-	-
31x31	-	-	-	-	-	-	-	-	-
32x32	-	-	-	-	-	-	-	-	-

(Tempos de execução medidos em segundos.)

de 3,8% a 262,9% em relação ao SatElite + Berkmin561. Apesar de perder em parte das instâncias para o NiVER + Berkmin561, o Vimplic + Berkmin561 é 20,1% mais rápido em média.

Finalmente, os resultados para as últimas instâncias de multiplicadores são apresentados. A Tabela 6.11 mostra os resultados para instâncias de CEC de multiplicadores *Dadda Tree*. Nessas instâncias, os tempos apresentados pelo Vimplic + Berkmin561 foram excelentes, principalmente, quando comparados com o Berkmin561 sozinho. A maior instância que o Berkmin561 consegue resolver no tempo limite é o multiplicador com operandos de 15 bits, enquanto o Vimplic + Berkmin561 resolve além de instâncias com operandos de 32 bits. Além disso, o aumento de desempenho varia de 440,1% a 1183,0%, ou seja, em alguns casos, mais de uma ordem de grandeza. Levando em consideração que o Berkmin561 é capaz de superar outros ótimos resolvedores como o Siege_v4, o Minisat e o Rsat para instâncias de CEC, os resultados para essas instâncias são certamente excelentes. O ganho de desempenho do Vimplic + Berkmin561 em relação ao C-SAT, ao NiVER + Berkmin561 e ao SatElite + Berkmin561 variam de 74,5% a 3302,9%, 4,4% a 121,0% e 1,1% a 316,7%, respectivamente. Note que o Vimplic + Berkmin561 foi o único capaz de resolver as instâncias de multiplicadores com operandos acima de 29 bits.

A Tabela 6.12 mostra o último conjunto de instâncias avaliadas que são proble-

Tabela 6.10: Resultados para Instâncias de CEC de Multiplicadores *Array* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
8x8	35,1	4,4	32,7	24,3	23,0	52,8	-80,8	42,3	5,9
9x9	50,3	30,1	47,7	95,8	26,4	90,5	14,1	80,6	262,9
10x10	79,3	214,9	109,9	110,6	61,6	28,8	249,2	78,5	79,7
11x11	172,3	1064,9	134,1	171,3	95,5	80,3	1014,8	40,4	79,3
12x12	217,7	6716,8	214,8	294,2	139,0	56,7	4733,6	54,6	111,7
13x13	338,9	>30000	301,1	402,1	198,3	70,9	-	51,8	102,8
14x14	546,5	-	471,3	543,9	365,8	49,4	-	28,8	48,7
15x15	872,1	-	807,1	832,6	504,6	72,8	-	60,0	65,0
16x16	1184,9	-	1028,0	1227,0	881,9	34,4	-	16,6	39,1
17x17	1882,9	-	1859,7	1738,7	1154,6	63,1	-	61,1	50,6
18x18	2399,7	-	1940,8	2593,4	1649,0	45,5	-	17,7	57,3
19x19	3410,4	-	2831,2	2795,6	2536,5	34,5	-	11,6	10,2
20x20	5452,5	-	3366,9	4307,4	3835,2	42,2	-	-12,2	12,3
21x21	6519,6	-	4712,1	5954,1	4544,2	43,5	-	3,7	31,0
22x22	8928,1	-	6176,5	8648,8	7003,3	27,5	-	-11,8	23,5
23x23	12322,2	-	8689,5	11178,9	10193,4	20,9	-	-14,8	9,7
24x24	15593,0	-	11633,7	12224,0	10758,7	44,9	-	8,1	13,6
25x25	19793,8	-	13207,3	16247,3	15651,1	26,5	-	-15,6	3,8
26x26	23934,2	-	17242,6	22266,5	19326,5	23,8	-	-10,8	15,2
27x27	29471,0	-	22632,8	26469,5	24072,2	22,4	-	-6,0	10,0
28x28	>30000	-	29521,8	>30000	29686,7	-	-	-0,6	-
29x29	-	-	>30000	-	>30000	-	-	-	-
30x30	-	-	-	-	-	-	-	-	-
31x31	-	-	-	-	-	-	-	-	-
32x32	-	-	-	-	-	-	-	-	-

(Tempos de execução medidos em segundos.)

mas de CEC derivadas de circuitos de divisores *Restoring*. Apesar de o Vimplic + Berkmin561 alcançarem a melhor colocação em oito dos dez conjuntos de instâncias já avaliados e a segunda melhor colocação nos outros dois conjuntos restantes, nas instâncias apresentadas na Tabela 6.12, pela primeira vez, o Vimplic + Berkmin561 ficou na terceira posição em relação ao tempo de resolução. Embora tenha superado o Berkmin561 e o C-SAT nessas instâncias fornecendo um ganho de desempenho variando de 1,4% a 240,2% e 1369,7% a 6056,8%, respectivamente, os resultados do NiVER + Berkmin561 e do SatElite + Berkmin561 foram melhores. O Vimplic + Berkmin561 foi de 15,1% a 60,8% mais lento do que o SatElite + Berkmin561 e de 2,2% a 58,5% mais lento do que o NiVER + Berkmin561.

Por meio das análises dos tempos de resolução apresentados nas Tabelas 6.2 a 6.12 é possível comprovar que o Vimplic + Berkmin561 é o conjunto de ferramentas mais rápido para resolver o benchmark selecionado. Além disso, em muitos casos, esse conjunto é o único que consegue resolver as instâncias maiores e, para a maior parte das instâncias e ferramentas, o ganho de desempenho percentual é maior do que 100%, podendo ultrapassar 1000% em alguns casos.

Além do tempo de resolução, outro fator essencial é o número de cláusulas produzidas após o pré-processamento pelas ferramentas selecionadas. A Tabela 6.13 mostra o percentual de acréscimo e de decréscimo no número de cláusulas após o

Tabela 6.11: Resultados para Instâncias de CEC de Multiplicadores *Dadda Tree* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
8x8	418,3	35,0	80,9	61,8	77,5	440,1	-54,8	4,4	-20,2
9x9	952,4	306,0	215,5	389,5	175,4	443,1	74,5	22,9	122,1
10x10	1826,3	1738,0	232,0	200,6	198,3	820,8	776,3	17,0	1,1
11x11	1946,1	11888,0	576,1	517,6	349,4	457,1	3302,9	64,9	48,2
12x12	4646,4	>30000	722,0	1062,8	593,2	683,3	-	21,7	79,2
13x13	9759,7	-	1732,0	2521,3	783,6	1145,5	-	121,0	221,8
14x14	10504,1	-	1740,6	3919,4	1895,8	454,1	-	-8,2	106,7
15x15	22582,0	-	2971,0	7335,0	1760,1	1183,0	-	68,8	316,7
16x16	>30000	-	3338,2	10332,5	4605,8	-	-	-27,5	124,3
17x17	-	-	4900,4	14342,8	4367,7	-	-	12,2	228,4
18x18	-	-	5783,9	10044,7	3938,0	-	-	46,9	155,1
19x19	-	-	7901,2	9397,2	4812,2	-	-	64,2	95,3
20x20	-	-	7332,4	11096,9	5318,7	-	-	37,9	108,6
21x21	-	-	10896,7	23428,5	9194,8	-	-	18,5	154,8
22x22	-	-	11989,3	18222,2	5876,4	-	-	104,0	210,1
23x23	-	-	18254,1	19416,0	7429,5	-	-	145,7	161,3
24x24	-	-	12261,6	15918,4	8383,0	-	-	46,3	89,9
25x25	-	-	15552,3	21963,9	12723,8	-	-	22,2	72,6
26x26	-	-	20124,6	20131,3	9795,3	-	-	105,5	105,5
27x27	-	-	20616,6	25940,0	13129,3	-	-	57,0	-
28x28	-	-	26179,2	23551,6	18818,8	-	-	39,1	-
29x29	-	-	24430,2	>30000	15542,8	-	-	57,2	-
30x30	-	-	>30000	-	24942,1	-	-	-	-
31x31	-	-	-	-	21705,1	-	-	-	-
32x32	-	-	-	-	21863,3	-	-	-	-
...	-	-	-	-	...	-	-	-	-

(Tempos de execução medidos em segundos.)

Tabela 6.12: Resultados para Instâncias de CEC de Divisores *Restoring* de Tamanhos Variados

	Tempos de Resolução de Instâncias de CEC em Segundos					*Speed-up* Percentual da VMPL+BK			
Tamanho	BK	C-SAT	NVR+BK	STL+BK	VMPL+BK	BK	C-SAT	NVR+BK	STL+BK
12 / 6	3,5	51,0	2,3	2,4	3,5	1,4	1369,7	-34,0	-31,4
14 / 7	16,6	336,0	12,0	8,0	9,8	69,0	3325,1	22,5	-18,4
16 / 8	37,4	1211,0	17,6	15,5	33,9	10,4	3474,4	-48,1	-54,3
18 / 9	59,5	5969,0	44,8	50,4	97,0	-38,6	6056,8	-53,8	-48,0
20 / 10	295,3	7200,0	45,5	62,1	93,0	217,6	7644,4	-51,0	-33,2
22 / 11	154,7	-	102,6	96,8	247,1	-37,4	-	-58,5	-60,8
24 / 12	782,0	-	118,8	137,3	229,8	240,2	-	-48,3	-40,3
26 / 13	425,6	-	231,3	508,6	432,5	-1,6	-	-46,5	17,6
28 / 14	614,0	-	414,2	257,4	567,8	8,1	-	-27,1	-54,7
30 / 15	1129,5	-	470,9	318,6	767,7	47,1	-	-38,7	-58,5
32 / 16	1829,0	-	716,3	561,8	1645,8	11,1	-	-56,5	-65,9
34 / 17	2732,9	-	1295,2	1204,6	1631,5	67,5	-	-20,6	-26,2
36 / 18	5574,0	-	3298,7	1035,2	2172,8	156,5	-	51,8	-52,4
38 / 19	7200,0	-	2965,2	1403,1	3782,2	90,4	-	-21,6	-62,9
40 / 20	-	-	3823,8	1666,6	3911,5	-	-	-2,2	-57,4
42 / 21	-	-	4919,0	3034,1	5587,0	-	-	-12,0	-45,7
44 / 22	-	-	5845,8	3892,8	6478,4	-	-	-9,8	-39,9
46 / 23	-	-	8634,4	4796,2	9038,9	-	-	-4,5	-46,9
48 / 24	-	-	7620,0	6404,5	8030,8	-	-	-5,1	-20,3
50 / 25	-	-	10596,4	9516,7	11211,2	-	-	-5,5	-15,1

(Tempos de execução medidos em segundos.)

Tabela 6.13: Acréscimo (+) ou Decréscimo (-) no número de cláusulas após o pré-processamento

Circuito	NiVER	SatELite	Vimplic
Somador *Carry Save*	-48%	-77%	+21%
Somador *Carry Skip*	-44%	-47%	+23%
Somador *Carry Select*	-48%	-52%	+125%
Somador *Block Carry LookAhead*	-38%	-44%	+30%
Somador *Ripple Carry Adder*	-48%	-49%	+22%
Multiplicador *Wallace Tree*	-28%	-52%	+401%
Multiplicador *Reduced Tree*	-52%	-56%	+146%
Multiplicador *CLA*	-26%	-29%	+270%
Multiplicador *Array*	-39%	-41%	+21%
Multiplicador *Dadda Tree*	-30%	-38%	+142%
Divisor *Restoring*	-66%	-54%	+32%

pré-processamento em relação o número total de cláusulas da instância original.

Primeiramente, pode-se notar que o SatElite consegue eliminar um número significativamente maior de cláusulas em 10 das 11 instâncias em relação ao NiVER. Esse fato já era esperado, pois, como explicado no Seção 4.2, o SatElite é uma extensão das técnicas de simplificação do NiVER. Além dessa observação pode-se perceber que, para todas as técnicas, o número de cláusulas resultantes varia muito de acordo com a classe do circuito.

Em relação ao Vimplic, a Tabela 6.13 mostra que, nas instâncias de multiplicadores *Carry LookAhead* e *Wallace Tree*, um conjunto bem grande de implicações (cláusulas) é derivado. Para o último multiplicador, a instância original do problema aumenta em 401%, esse acréscimo provavelmente iria reduzir significativamente o desempenho do resolvedor de SAT. Todavia, como as implicações adicionadas pela Vimplic são cláusulas binárias, sempre que algum dos literais dessa cláusula adicionada é selecionado para decisão, diversas cláusulas unitárias são geradas. Como o mecanismo de BCP é bastante eficiente, tais cláusulas, mesmo em excesso, auxiliam bastante a redução do tempo de resolução do resolvedor de SAT como mostrado nas Tabelas 6.7 e 6.9.

Além das considerações de aumento no número de cláusulas, a avaliação da redução do número de cláusulas também é importante. A ferramenta NiVER foi capaz de reduzir o número de cláusulas de 26% a 66% em relação ao tamanho da instância original, porém, em grande parte dos casos, o tempo de utilizado pelo resolvedor aumentou consideravelmente. Isso prova que reduzir o tamanho da instância não implica em uma redução no tempo de resolução e ainda que, em muitos casos, a redução pode, na realidade, aumentar o tempo de resolução. Além disso, o aumento do número de cláusulas (informações redundantes), como feito pela ferramenta Vimplic, muitas vezes, reduz o tempo de resolução criando uma fórmula maior. Finalmente, é importante destacar que essas observações são similares e estão coerentes com as conclusões já apresentadas

nos trabalhos da área de Satisfabilidade [BR00][ZM06][FGMS07].

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

O presente trabalho apresentou duas importantes contribuições para o problema de Verificação de Equivalência de Circuitos Combinacionais (CEC, do Inglês, *Combinational Equivalence Checking*). A primeira contribuição resultou em quatro outras contribuições menores, já a segunda resultou em três menores. Essa primeira contribuição é uma técnica de pré-processamento que deriva informações redundantes dos dois circuitos sob CEC de modo a reduzir o tempo utilizado pelo Resolvedor de Satisfabilidade (SAT) para a prova de equivalência entre ambos circuitos. Através dessa técnica, é possível superar em desempenho as principais técnicas do estado da arte na CEC baseado em SAT.

As quatro contribuições menores relativas à técnica de pré-processamento são as seguintes. A primeira delas e a mais importante é o estabelecimento de relações entre o presente trabalho e os trabalhos na área de Satisfabilidade através do estudo de redundância em fórmulas em Forma Normal Conjuntiva (CNF, do Inglês, *Conjunctive Normal Form*). Através da aplicação, estudo e análise dos resultados obtidos com a ferramenta Vimplic, foi possível apresentar observações e conclusões que são coerentes com as já apresentadas nos trabalhos da área de Satisfabilidade. É importante destacar que essas conclusões obtidas poderiam ser divergentes das conclusões dos trabalhos de Satisfabilidade devido à utilização de diferentes classes de problemas, benchmarks, técnicas para conversão para CNF, derivação ou eliminação de cláusulas redundantes, dentre outros. A segunda contribuição menor é a análise de como diferentes ferramentas de pré-processamento comportam-se em instâncias de CEC. Isso será útil para ajudar os futuros desenvolvedores de ferramentas de pré-processamento a selecionar as mais adequadas tanto para estabelecer comparativos quanto para a sua utilização. A terceira contribuição menor é a demonstração de que fórmulas menores em CNF não implicam necessariamente em problemas mais fáceis para o resolvedor de SAT, principalmente em

instâncias de CEC. Em muitos casos, as fórmulas originais consumiam menos tempo de resolução do que a fórmula menor pré-processada. A quarta contribuição menor é a discussão detalhada de como ferramentas de derivação de implicações podem ser implementadas de modo bastante eficiente.

A segunda contribuição do presente trabalho é uma ferramenta para geração de circuitos, a BenCGen, que tem como principal objetivo a produção de circuitos para benchmarks. Essa ferramenta é capaz de gerar 24 tipos de circuitos diferentes com tamanhos parametrizados. Variando-se do menor para o maior tamanho de cada circuito, mais de 1.000.000 circuitos podem ser gerados. Tal ferramenta vem suprir uma grande necessidade de novos benchmarks para CEC e para outras áreas de Verificação Formal.

As três contribuições menores relativas à BenCGen são as seguintes. A primeira e a mais importante é a definição do resolvedor de SAT mais adequado para problemas de CEC. Através da análise feita foi possível constatar que, apesar de muitos pesquisadores utilizarem a Competição internacional de SAT para escolherem um resolvedor como referência, essa escolha pode levar a conclusões erradas em relação ao desempenho do mesmo, principalmente na área de CEC. A segunda contribuição menor é mostrar que em algumas publicações recentes sobre CEC em conferências importantes de Automação de Projetos Eletrônicos (EDA, do Inglês, *Electronic Design Automation*), o Minisat é utilizado como referência para problemas de verificação de equivalência através de resolvedores de SAT. Isso é feito sem nenhuma definição de critério de seleção de resolvedores. Como discutido no presente trabalho, dependendo do tipo de instâncias analisadas, uma comparação com o Minisat pode ser considerada pouco justa, principalmente se o objetivo for mostrar que a nova técnica apresentada supera esse resolvedor. A terceira contribuição menor é a análise de todos os principais benchmarks para a área de verificação formal destacando os seus principais benefícios e limitações.

Finalmente, como resultado das duas principais contribuições, três ferramentas foram elaboradas e todas elas podem ser solicitadas pelo e-mail *vivas@decom.cefetmg.br*. A primeira delas é uma ferramenta que cria um circuito *miter* a partir de dois outros, converte-o para CNF, cria uma propriedade egarante que a saída do circuito gerado não pode ser assinalada com 1. A segunda ferramenta, a Vimplic, é o resultado da implementação da técnica de pré-processamento proposta. A terceira ferramenta, a BenCGen, é a de geração de circuitos combinacionais.

7.2 Trabalhos Futuros

Os trabalhos futuros descritos nessa seção abordam continuidades do presente trabalho e a elaboração de outros trabalhos distintos, mas relacionados com o tema. Primeira-

mente, são discutidas as continuidades ou extensões do trabalho em ordem crescente de dificuldade.

As seguintes extensões podem ser destacadas:

- Aumentar a eficiência da ferramenta de criação do circuito *miter* para reduzir o tempo gasto na sua produção. Apesar de essa economia de tempo não resultar em nenhum impacto nos tempos apresentados no presente trabalho, já que as instâncias do problema são sempre geradas antes de se elaborar os testes.

- Aumentar os formatos de entrada aceitos pela ferramenta Vimplic. Atualmente, a ferramenta suporta apenas o formato BENCH que é o formato mais freqüentemente aceito pelas ferramentas de CEC.

- Em relação à ferramenta Vimplic, novas estruturas de dados mais eficientes podem ser estudadas e implementadas com o intuito de reduzir o tempo de criação, armazenamento e derivação de cláusulas por meio do grafo de implicações. Isso possibilitaria a aplicação dessa ferramenta em circuitos bem pequenos, o que não é viável atualmente.

- Novos tipos de heurísticas como a detecção de sinais equivalentes e sinais constantes podem ser incluídas na ferramenta Vimplic. Isso pode resultar na derivação de cláusulas que auxiliarão a reduzir ainda mais o tempo de verificação. No entanto, nas versões preliminares da ferramenta, ambas as abordagens foram testadas sem sucesso, pois o tempo utilizado por essas heurísticas era grande e o benefício era bastante limitado.

Em relação a trabalhos futuros mais gerais, os seguintes podem ser destacados:

- A ferramenta BenCGen é capaz de gerar diversos circuitos combinacionais. No entanto, circuitos seqüenciais ainda não foram contemplados. Existem diversos circuitos aritméticos seqüenciais, unidades controladoras de processadores, contadores síncronos e assíncronos que podem ser implementados. Essa extensão resultaria na ferramenta mais completa de geração de circuitos parametrizados publicada até o momento.

- Além de se implementar novos circuitos para a BenCGen, também é necessário analisar as características desses circuitos gerados após submetê-los a um processo de síntese. Certamente, esse não era o foco do presente trabalho. Todavia, se bons resultados de área e freqüência forem obtidos na síntese, a ferramenta poderá ser utilizada, além da geração de circuitos para benchmarks, para a produção de núcleos de propriedade intelectual para os desenvolvedores de circuitos integrados.

- A ferramenta Vimplic, apresentada no presente trabalho, é utilizada para derivar cláusulas redundantes de circuitos combinacionais. Essa ferramenta pode ser estendida para derivar cláusulas redundantes de circuitos seqüenciais de modo a aumentar a eficiência dos resolvedores de Verificação de Equivalência de Circuitos Seqüenciais. Para isso, novas análises e estudos devem ser conduzidos nesses tipos de circuitos.

- Outro trabalho interessante que pode ser elaborado é a criação de um resolvedor de SAT para CEC em código aberto. Certamente, a viabilização dessa tarefa exige pesquisas e análises bastante detalhadas a respeito dos resolvedores de SAT atuais. O Berkmin561, por meio da sua estratégia especial para CEC, poderia suprir essa necessidade, entretanto, dois problemas dificultam a sua utilização. O primeiro problema é que apenas versões executáveis são disponibilizadas e, por isso, detalhes de implementação não são conhecidos. Apenas características superficiais da sua implementação são descritas nas publicações relativas a esse resolvedor. O segundo problema é que o Berkmin561 é uma ferramenta proprietária e a sua utilização depende da aceitação do termo de licença de uma empresa.

- Um novo trabalho também pode ser desenvolvido com o intuito de se analisar a relação entre a ordem em que as cláusulas aparecem em instâncias de SAT e a eficiência do resolvedor. Note que, no presente trabalho, as cláusulas binárias derivadas sempre são adicionadas ao final do arquivo (ou fórmula) da instância original. Em grande parte dos resolvedores, diferentes resultados poderiam ser encontrados se as cláusulas fossem adicionadas em outra parte do arquivo original. Essa característica está relacionada com o mecanismo de decisão utilizado pelo resolvedor de SAT escolhido. No entanto, nos testes feitos no presente trabalho, a adição das cláusulas no início ou no fim do arquivo da instância original não resultou em uma diferença significativa no desempenho.

- Na área de Satisfabilidade, uma das grandes questões é a de como avaliar a dificuldade de resolução de uma instância de acordo com as suas características tais como o tamanho da fórmula, o número de cláusulas e literais, os tipos de cláusulas, dentre outros. Já existem medidas seguras para a avaliação da dificuldade de resolução de fórmulas aleatórias em CNF. Contudo, até o momento, não existe nenhuma medida segura para avaliar a dificuldade de fórmulas de problemas industriais gerais e nem de fórmulas de problemas industriais específicos de cada área. Note que a elaboração de um trabalho como esse, resultaria em uma enorme contribuição tanto teórica quanto prática para as comunidades de verificação formal, otimização e satisfabilidade, dentre outras.

Apêndice A

Lista de Siglas

AMG - *Arithmetic Module Generator*
ATPG - *Automatic Test Pattern Generation*
BCP - *Boolean Constraint Propagation*
BDD - *Binary Decision Diagram*
BLIF - *Berkeley Logic Interchange Format*
BMD - *Binary Moment Diagram*
CEC - *Combinational Equivalence Checking*
CNF - *Conjunctive Normal Form*
EDA - *Electronic Design Automation*
FPGA - *Field-Programmable Gate Arrays*
ISCAS - *IEEE International Symposium on Circuits and Systems*
ITC - *International Test Conference*
MOM - *Maximum Occurrences on Clauses of Minimum Sizes*
PHDD - *Multiplicative Power Hybrid Decision Diagrams*
RCUL - *Regra da Cláusula de Um Literal*
RL - *Recursive Learning*
ROBDD - *Reduced Ordered Binary Decision Diagram*
RTL - *Register Transfer Level*
SAT - *Satisfabilidade*
TFI - *Transitive Fan-In*
ULA - *Unidade Lógica e Aritmética*
UP - *Unit Propagation*
VER - *Variable Elimination Resolution*
VLIW - *Very-Long Instruction Word*
VSIDS - *Variable State Independent Decaying Sum*

Referências Bibliográficas

[AH04] Rajat Arora and Michael S. Hsiao. Using global structural relationships of signals to accelerate SAT-based combinational equivalence checking. *Journal of Universal Computer Science*, 10(12):1597–1628, 2004.

[Alb05] Christoph Albrecht. International Workshop on Logic and Synthesis 2005 benchmarks. Disponível em http://iwls.org/iwls2005/benchmarks.html, 2005.

[Alt08] Altera. Quartus II Megawizard Plug-in Manager. Disponível em http://www.altera.com/products/ip/, 2008.

[AOF07] Fabrício Vivas Andrade, Márcia C. M. Oliveira, and Antônio Otávio Fernandes. SAT-based equivalence checking based on circuit partitioning and special approaches for conflict clause reuse. In *Proceedings of IEEE Workshop on Design and Diagnosis of Electronic Circuits and Systems*, pages 397–402, April 2007.

[ASF08a] Fabricio Vivas Andrade, Leandro Maia Silva, and Antonio Otavio Fernandes. BenCGen: A digital circuit generation tool for benchmarks. In *Proceedings of ACM Symposium on Integrated Circuits and Systems Design*, pages 164–169, September 2008.

[ASF08b] Fabricio Vivas Andrade, Leandro Maia Silva, and Antonio Otavio Fernandes. SAT-based combinational equivalence checking through circuit preprocessing. In *Proceedings of IEEE International Conference on Computer Design*, pages 40–45, October 2008.

[BAA+01] D. Bakalis, K. D. Adaos, G. Ph. Alexiou, D. Nikolos, and D. Lymperopoulos. Eudoxus: A www-based generator of reusable arithmetic cores. In *Proceedings of the 12th International Workshop on Rapid System Prototyping*, pages 182–193, Washington, DC, USA, 2001. IEEE Computer Society.

[BBN+04] Bob Bentley, Kurt Baty, Kevin Normoyle, Makoto Ishii, and Einat Yogev. Verification: what works and what doesn't. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 274–274, June 2004.

[BC95] Randal E. Bryant and Yirng-An Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 535–541, New York, NY, USA, 1995. ACM Press.

[Bei95] B. Beizer. The Pentium Bug - an Industry Watershed. Testing Techniques Newsletter, TTN, On-line edition, December 1995.

[Ber05] Berkeley Logic Synthesis and Verification Group. ABC: A system for sequential synthesis and verification, release 51205. http://www-cad.eecs.berkeley.edu/∼alanmi/abc/, 2005.

[BF85] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran. In *Proceedings of the International Symposium on Circuits and Systems*, pages 663–698, June 1985.

[BR00] Yacine Boufkhad and Olivier Roussel. Redundancy in random SAT formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 273–278, 2000.

[Bra93] Daniel Brand. Verification of large synthesized designs. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 534–537, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[Bry91] Randal E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[BS98] Jerry R. Burch and Vigyan Singhal. Tight integration of combinational verification methods. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 570–576, New York, NY, USA, 1998. ACM Press.

[Bur91] Jerry R. Burch. Using BDDs to verify multipliers. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 408–412, New York, NY, USA, 1991. ACM Press.

[BW03] F. Bacchus and J. Winter. Effective preprocessing with hyper-resolution and equality reduction. In *Theory and Applications of Satisfiability Testing*, pages 341–355, 2003.

[CC01] Jiunn-Chern Chen and Yirng-An Chen. Equivalence checking of integer multipliers. In *Proceedings of the Asia South Pacific Design Automation Conference*, pages 169–174, New York, NY, USA, 2001. ACM Press.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[CRS00] F. Corno, M. Sonza Reorda, and G. Squillero. RT-level ITC 99 benchmarks and first ATPG results. *IEEE Design and Test of Computers*, pages 44–53, July-August 2000.

[Dav99] Scott Davidson. ITC'99 benchmark circuits - preliminary results. Benchmarks disponíveis em http://www.cerc.utexas.edu/itc99-benchmarks /bench.html, 1999.

[DD96] Joel Darnauer and Wayne Wei-Ming Dai. A method for generating random circuits and its application to routability measurement. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*, pages 66–72, New York, NY, USA, 1996. ACM.

[DK03] Robert Damiano and James Kukula. Checking satisfiability of a conjunction of BDDs. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 818–823, New York, NY, USA, 2003. ACM Press.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the Association for Computer Machinery*, 5(7):394–397, July 1962.

[DP60] M. Davis and H. Putnam. A computation procedure for quantification theory. *Journal of the Association for Computer Machinery*, 7:201–215, 1960.

[Dre04] Rolf Drechsler. *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *International Conference on Theory and Applications of Satisfiability Testing,*, pages 61–75, 2005.

[ES03] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 840–843, June 2003.

[FBK89] D. Bryan F. Brglez and K. Kozminski. Combinational problems of sequential benchmark circuits. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1929–1934, June 1989.

[FFK88] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on Binary Decision Diagrams. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 2–5, November 1988.

[FGMS07] Olivier Fourdrinoy, Eric Gregoire, Bertrand Mazure, and Lakhdar Sais. Eliminating redundant clauses in SAT instances. In *The International Conference on Integration of AI and OR Techniques in Techniques in Constraint Programming for Combinatorial Optimisation Problem*, pages 71–83, 2007.

[FKL04] Harry Foster, Adam C. Krolnik, and David J. Lacey. *Assertion-Based Design*. Kluwer Academic Publishers, 2004.

[GN02] E. Goldberg and Y. Novikov. Berkmin: A fast and robust SAT-solver. In *Proceedings of the Conference on Design, Automation and Test in*

Europe, pages 142–149, Washington, DC, USA, 2002. IEEE Computer Society.

[GN03] E. Goldberg and Y. Novikov. Equivalence checking of dissimilar circuits. In *Workshop Notes of The International Workshop on Logic Synthesis*, May 2003.

[GPB01] E. Goldberg, M. Prasad, and R. Brayton. Using SAT for combinational equivalence checking. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 114–121, Piscataway, NJ, USA, 2001. IEEE Press.

[Har00] Justin E. Harlow. Overview of popular benchmark sets. *IEEE Design and Test of Computers*, 17(3):15–17, 2000.

[HMY95] Kiyoharu Hamaguchi, Akihito Morita, and Shuzo Yajima. Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 78–82, Washington, DC, USA, 1995. IEEE Computer Society.

[HWAH06] Naofumi Homma, Yuki Watanabe, Takafumi Aoki, and Tatsuo Higuchi. Formal design of arithmetic circuits based on arithmetic description language. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(12):3500–3509, 2006.

[JW90] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, pages 167–187, 1990.

[KDB+97] Martin Keim, Rolf Drechsler, Bernd Becker, Michael Martin, and Paul Molitor. Polynomial formal verification of multipliers. In *Proceedings of VLSI Test Symposium*, pages 150–155, 1997.

[KGP01] Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based Boolean reasoning. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 232–237, New York, NY, USA, 2001. ACM Press.

[KK97] Andreas Kuehlmann and Florian Krohm. Equivalence checking using cuts and heaps. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 263–268, 1997.

[KP93] Wolfgang Kunz and Dhiraj K. Pradhan. Accelerated dynamic learning for test pattern generation in combinational circuits. *IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems*, pages 684–693, May 1993.

[KP94] Wolfgang Kunz and Dhiraj K. Pradhan. Recursive learning: A new implication technique for efficient solutions to CAD-problems: Test, verification and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1143–1158, September 1994.

[Kun93] Wolfgang Kunz. Hannibal: an efficient tool for logic verification based on recursive learning. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 538–543, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of the Conferece of Principles and Practice of Constraint Programming*, pages 341–355, 1997.

[Lar92] Tracy Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11(1):6–22, 1992.

[LWC+03] Feng Lu, Li-C. Wang, K.-T. (Tim) Cheng, John Moondanos, and Ziyad Hanna. A signal correlation guided ATPG solver and its applications for solving difficult industrial cases. pages 436–441, New York, NY, USA, 2003. ACM Press.

[MCBE06] Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Niklas Een. Improvements to combinational equivalence checking. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 836–843, New York, NY, USA, 2006. ACM.

[MD04] Prabhat Mishra and Nikil Dutt. Funcional validation of programmable architectures. In *Proceedings of the EUROMICRO Systems on Digital System Design*, pages 12–19, 2004.

[MHC02] J. Rose M.D. Hutton and D. Corneil. Automatic generation of synthetic sequential benchmark circuits. *IEEE Transations on Computer-Aided Design of Integrated Circuits and Systems*, 21(8):928–940, 2002.

[MJT+97] R. Mukherjee, J. Jain, K. Takayama, M. Fujita, J. A. Abraham, and D. S. Fussell. Flover: Filtering oriented combinational verification approach. In *Workshop Notes of International Workshop on Logic Synthesis*, 1997.

[MM04] Paul Molitor and Janett Mohnke. *Equivalence Checking of Digital Circuits - Fundamentals, Principles, Methodos*. Kluwer Academic Publishers, 2004.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: engineering an efficient SAT solver. pages 530–535, New York, NY, USA, 2001. ACM Press.

[Moo65] Gordon Moore. Cramming more componentes onto integrated circuits. *Electronics*, 38(8), 1965.

[MWBSV88] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using Binary Decision Diagrams in a logic synthesis environment. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 6–10, 1988.

[Ope08] OpenCores Website. Disponível em http://www.opencores.org/, 2008.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. Rsat 2.0: Sat solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

[PLM99] J. Pistorius, E. Legaiand, and M. Minoux. Generation of very large circuits to benchmark the partitioning of FPGA. In *Proceedings of the International Symposium on Physical Design*, pages 67–73, New York, NY, USA, 1999. ACM Press.

[Rud93] R. Rudell. Dynamic variable ordering for Ordered Binary Decision Diagrams. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[RWS00] S. Reda, A. Wahba, and A. Salem. M-check: A multiple engine combinational equivalence checker. In *Proceedings of the International Symposium on Circuits and Systems*, pages 613–616, May 2000.

[Rya04] Lawrence Ryan. *Efficient algorithms for clause-learning SAT solvers.* PhD thesis, Simon Fraser University, School of Computing Science, February 2004.

[Sch03] Tom Schubert. High level formal verification of next-generation microprocessors. In *Proceedings of ACM/IEEE Conference on Design Automation*, pages 1–6, 2003.

[Sil95] João P. Marques Silva. *Search algorithms for satifiability problems in combinational switching circuits.* PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, May 1995.

[Sil99] J. Marques Silva. The impact of branching heuristics in propositional satisfiability algorithms. In *Proceedings of the Portuguese Conference on Artificial Inteligence*, pages 62–74, September 1999.

[Sil00] João P. Marques Silva. Algebraic simplification techniques for propositional satisfiability. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, pages 537–542, 2000.

[SK04] Dominik Stoffel and Wolfgang Kunz. Equivalence checking of arithmetic circuits on the arithmetic bit level. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(5):586–597, 2004.

[SP04] S. Subbarayan and Dhiraj Pradhan. NiVER: Non increasing variable elimination resolution. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, pages 351–356, 2004.

[SS96] João P. Marques Silva and Karem A. Sakallah. Grasp - a new search algorithm for satisfiability. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, Washington, DC, USA, 1996. IEEE Computer Society.

[SS99] J. Marques Silva and L. G. Silva. Algorithms for satisfiability in combinational circuits based on backtrack search and recursive learning. In *Workshop Notes of the International Workshop on Logic Synthesis*, pages 227–241, June 1999.

[STS88] M. H. Schulz, E. Trischler, and T. M. Sarfen. Socrates: A highly efficient Automatic Test Pattern Generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7:126–137, January 1988.

[TGA00] Paul Tafertshofer, Andreas Ganz, and Kurt Antreich. IGRAINE-an implication graph-based engine for fast implication, justification, and propagation. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 19(8):907–927, 2000.

[Tse83] G. S. Tseitin. On the complexity of derivation in propositional calculus. In J. Siekmann and G. Wrightson, editors, *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, pages 466–483. Springer, Berlin, Heidelberg, 1983.

[vEJ94] C. A. J. van Eijk and G. L. J. M. Janssen. Exploiting structural similarities in a BDD-based verification method. In *Proceedings of the International Conference on Theorem Provers in Circuit Design*, volume 901, pages 110–125. Springer-Verlag, 1994.

[Vel04] Miroslav N. Velev. A new generation of ISCAS benchmarks from formal verification of high-level microprocessors. In *Proceedings International Symposium on Circuits and Systems*, pages 213–216, 2004.

[VSC02] P. Verplaetse, D. Stroobandt, and J. Van Campenhout. Synthetic benchmark circuits for timing-driven physical design applications. In *Proceedings of the International Conference on VLSI*, pages 31–37, Las Vegas, Nevada, USA, 2002. CSREA Press.

[WLLH07] Chi-An Wu, Ting-Hao Lin, Chih-Chun Lee, and Chung-Yang (Ric) Huang. QuteSAT: a robust circuit-based SAT solver for complex circuit structure. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1313–1318, San Jose, CA, USA, 2007.

[wp08] The International SAT Competitions web page. Disponível em http://www.satcompetition.org/, 2008.

[XYLG03] Zhan Xu, Xiaolang Yan, Yongjiang Lu, and Haitong Ge. Equivalence checking using independent cuts. In *Proceedings of Asian Test Symposium*, page 482, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[ZM06] H. Zeng and S. McIlraith. Experimental results on the satisfiable core in random 3SAT. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, 2006.

[ZNP97] J. Zhao, J. A. Newquist, and J. Patel. Static logic implication with application to fast redundancy identification. In *Proceedings of VLSI Test Symposium*, pages 288–293, 1997.