VERIFICAÇÃO DE EQUIVALÊNCIA COMBINACIONAL UTILIZANDO HIPER-RESOLUÇÃO BINÁRIA

LEONARDO VASCONCELOS ALVES

VERIFICAÇÃO DE EQUIVALÊNCIA COMBINACIONAL UTILIZANDO HIPER-RESOLUÇÃO BINÁRIA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Antônio Otávio Fernandes

Belo Horizonte

31 de agosto de 2010

Alves, Leonardo Vasconcelos
A474v Verificação de Equivalência Combinacional utilizando hiper-resolução binária / Leonardo Vasconcelos Alves. — Belo Horizonte, 2010
xxviii, 86 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Antônio Otávio Fernandes

1. Circuitos integrados - Tese. 2. Circuitos digitais - Tese. 3. Combinatória - Tese. I. Título.

CDU 519.6*17 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Verificação de equivalência combinacional utilizando hiper-resolução binária

LEONARDO VASCONCELOS ALVES

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. ANTÔNIO OTÁVIO FERNANDES - Orientador
Departamento de Ciência da Computação - UFMG

PROF. ROMANELLI LODRON ZUIM
Departamento de Ciência da Computação - PUC

PROF. DIÓGENES CECÍLIO DA SILVA JÚNIOR
Departamento de Engenharia Elétrica - UFMG

Belo Horizonte, 31 de agosto de 2010.

Dedico este trabalho a Deus, à N.S. Aparecida, à minha família, à Alline e aos meus amigos.

Agradecimentos

Agradeço a Deus Pai por ter me dado todas as condições necessárias para chegar até aqui e por ter me ajudado mesmo quando estive totalmente desanimado. Agradeço a Deus Filho por me mostrar o caminho correto e por sua infinita misericórdia, sempre me acolhendo, mesmo quando saio do caminho. Agradeço ao Espírito Santo por mover todas as coisas para o bem e por me iluminar nos mais difíceis momentos. Agradeço a proteção de N.S. Aparecida, S. Antônio, S. Expedito, e de todos os outros santos a quem recorri, recorro e sempre recorrerei. Agradeço também ao Pe. Libério, que, tenho certeza, nos ajuda nesta caminhada.

Agradeço aos meus pais, irmãos, tios e avó por confiarem na minha capacidade e me ajudarem sempre. Agradeço ainda à Alline, minha namorada, por estar ao meu lado durante todo o período do curso de mestrado, sempre proferindo palavras de ânimo e otimismo, fornecendo amor, carinho e paciência. Agradeço aos grandes amigos Paulo e Daniel, caras com quem posso contar a qualquer momento, para qualquer assunto. Agradeço à Tilly e à D. Lurdinha por me acolherem tão bem nesses últimos quatro anos.

Agradeço também ao Antônio Otávio e ao Fabrício Vivas pela paciência e confiança depositada durante as várias desventuras nos últimos anos. Ao Makish pelas várias dicas e por auxiliar na revisão do texto. Obrigado também ao Valdeci, Leandro Maia, Gabriel Cysne e Paulinho, pessoas que me deram grandes oportunidades de crescimento profissional enquanto trabalhei no projeto Scanitec. Agradeço a todo o pessoal do Lecom, pelos momentos de descontração e companhia.

Por fim, não posso deixar de agradecer a todo o pessoal do Departamento de Ciência da Computação, em especial à secretaria, por toda a infra-estrutura e suporte fornecidos, tanto como aluno quanto como professor.

"Você não tem uma alma. Você é uma Alma.
Você tem um corpo."
(C.S. Lewis)

Resumo

A capacidade de integrar cada vez mais componentes dentro de circuitos integrados tem dobrado a aproximadamente cada 18 meses desde meados de 1960, com previsões de continuar neste ritmo até 2050. Esta crescente complexidade de dispositivos computacionais leva a indústria de circuitos integrados a investir mais de 50% do tempo de desenvolvimento na etapa de verificação dos novos projetos, buscando minimizar prejuízos decorrentes da substituição de componentes defeituosos. Este trabalho propõe melhorar o processo de verificação de circuitos digitais combinacionais utilizando pré-processamento baseado em técnicas de hiper-resolução binária, aplicadas em grafos de implicações extraídas dos próprios circuitos. O texto descreve uma ferramenta que implementa estas técnicas; ao final são analisados os resultados obtidos que comprovam a eficiência da hiper-resolução binária como técnica de simplificação, mas que ao mesmo tempo eliminam quaisquer informações estruturais referentes ao circuito sob verificação.

Abstract

The capacity of cramming more components into integrated circuits has doubled roughly every 18 months since mid-1960, with projections to continue this pace until 2050. This growing complexity of computing devices leads the integrated circuit industry to invest more than 50% of development time in the stage of verification of new projects in order to minimize losses due to the replacement of defective components. This paper proposes an improvement for the combinational equivalence checking problem using pre-processing techniques based on hyper binary resolution, applied to graphs extracted from circuit implications. The text describes a tool that implements these techniques, and in the end the results shows the efficacy of hyper-binary resolution as a technique for simplification, but at the same time eliminate any structural information concerning the circuit under verification.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Listagens de códigos-fonte

Sumário

Capítulo 1

Introdução

Este trabalho tem como objetivo contribuir para a solução do problema da verificação de equivalência entre circuitos combinatórios (CEC, do inglês *Combinatory Equivalence Checking*) analisando o impacto da técnica de hiper-resolução binária aplicada no pré-processamento de circuitos sob verificação.

De forma geral, o processo de verificação de um produto consiste em averiguar se suas características estão de acordo com as especificações do seu projeto. Em se tratando da indústria de circuitos integrados, tal verificação é de grande importância: se um circuito entrar no mercado com algum problema no seu funcionamento podem ocorrer grandes prejuízos financeiros e até mesmo a eliminação daquela empresa do mercado, tal o prejuízo em sua imagem. No caso de circuitos eletrônicos, um dos processos mais comuns para verificação é o chamado de verificação de equivalência. A verificação de equivalência é uma técnica formal que consiste em analisar as características de um circuito sob análise, comparando com um circuito de referência em busca de eventuais diferenças. Caso não hajam diferenças entre os circuitos é dito que os são circuitos equivalentes.

O foco deste trabalho está na verificação de equivalência de circuitos combinatórios, problema já declarado NP-difícil por vários autores [Ibarra & Sahni, 1975; Keim et al., 2003; Mishchenko et al., 2006]. Apesar de tal dificuldade, várias técnicas foram propostas para deixá-lo mais tratável: Bryant [1986] mostra que diagramas de decisão binária e diagramas de momento binário [Bryant & Chen, 1995] são capazes de verificar alguns circuitos lógicos, mas falham ao verificar multiplicadores; Brand [1993] introduz uma técnica de verificação baseada em geração automática de padrões de teste e, mais recentemente, resolvedores de satisfabilidade booleana (SAT) têm sido utilizados com sucesso para executar este tipo verificação [Larrabee, 1992; Goldberg et al., 2001; Moskewicz et al., 2001; Goldberg & Novikov, 2002; Eén & Sörensson, 2003; Prasad

et al., 2005; Oliveira, 2006; Andrade et al., 2007; Mendes, 2008]. O grande problema com a maioria dos resolvedores SAT é que, para executar a verificação de equivalência combinatória, informações estruturais a respeito dos circuitos são perdidas.

Para tentar contornar o problema de perda de informações estruturais, trabalhos como os de Lu et al. [2004], Arora & Hsiao [2004] e Andrade et al. [2008b] trabalham diretamente com circuitos eletrônicos. Enquanto o primeiro é um resolvedor SAT para circuitos (C-SAT) os últimos calculam implicações extraídas de circuitos digitais combinatórios, gerando novas cláusulas a serem adicionadas no problema de verificação original para posteriormente serem aplicadas em um resolvedor SAT baseado em CNF, obtendo bons resultados.

Paralelamente, Bacchus & Winter [2003] e Gershman & Strichman [2005] desenvolveram seus trabalhos baseando em técnicas de hiper-resolução propostas por Robinson [1965] para pré-processar fórmulas booleanas na forma normal conjuntiva e alimentar resolvedores SAT. A hiper-resolução permite efetuar grandes simplificações no problema original, reduzindo a instância aplicada aos resolvedores SAT e, em conseqüência, reduzir o tempo de execução.

1.1 Objetivos

O presente trabalho tem por objetivo analisar o impacto da técnica de hiper-resolução binária aplicada no pré-processamento de implicações que podem ser derivadas de circuitos digitais combinatórios para verificação de equivalência combinatóra. Para tal, serão geradas implicações a partir de cada porta lógica do circuito e também através da técnica proposta por Andrade et al. [2008b]. Estas implicações serão então processadas por um hiper-resolvedor binário como o proposto por Bacchus & Winter [2003] e, ao fim do pré-processamento, ou serão obtidas cláusulas na forma normal conjuntiva (CNF) que podem ser aplicadas em resolvedores SAT, ou o próprio hiper-resolvedor binário poderá encontrar conflitos no circuito sob verificação e não demandar a execução do resolvedor SAT.

1.2 Motivação do trabalho

Desde simples controles remotos a computadores de grande porte responsáveis por processamento de modelos matemáticos que permitem obter a previsão do tempo, os circuitos eletrônicos permeiam nossa vida. Propagandas mostram a todo momento as últimas novidades em *smartphones*, leitores de livros digitais e PDA's, cada um com

novas aplicações, demandando capacidades de processamento cada vez maiores e com consumo de energia cada vez menor.

Fato é que a capacidade de integrar cada vez mais componentes dentro de circuitos integrados tem dobrado a aproximadamente cada 18 meses desde meados de 1960, com previsões de continuar neste ritmo até 2050 [Moore, 1965; Schaller, 1997; Tanenbaum, 2001]. Graças a essa grande capacidade de integração, circuitos integrados digitais cada vez mais complexos têm sido elaborados para atender às mais variadas áreas de aplicação dos sistemas de computação.

Atualmente, a grande complexidade dos dispositivos leva a indústria de circuitos integrados a investir mais de 50% do tempo de desenvolvimento na etapa de verificação dos novos projetos [Bentley et al., 2004; Foster et al., 2004; Drechsler, 2004], buscando minimizar novas ocorrências como o bug do Pentium[1], descoberto em 1994, que causou um prejuízo de quase um bilhão de dólares à Intel Corporation [Beizer, 1995].

O trabalho de verificação, portanto, influencia diretamente o preço e tempo de lançamento do produto no mercado, com o objetivo de garantir a qualidade do circuito integrado. A motivação deste trabalho é contribuir exatamente nesta área de projeto e desenvolvimento de circuitos integrados, buscando desenvolver uma ferramenta capaz de executar, em tempo hábil, verificações de circuitos aplicados nos sistemas computacionais atuais e que seja capaz de manter informações estruturais do circuito sob verificação.

1.3 Organização do texto

Esta dissertação está organizada em 5 capítulos. O capítulo 1 introduz o tema da dissertação. O capítulo 2 mostra uma revisão bibliográfica sobre o problema de verificação de equivalência combinatória e hiper-resolução binária. O capítulo 3 mostra o desenvolvimento do trabalho, mostrando algoritmos e metodologias implementadas. O capítulo 4 descreve os resultados obtidos neste trabalho e o capítulo 5 mostra as conclusões obtidas.

[1] Marca registrada da Intel Corporation.

Capítulo 2

Revisão bibliográfica

Este capítulo apresenta definições básicas importantes para a compreensão do problema de verificação de equivalência combinatória, uma revisão bibliográfica sobre o assunto e descreve alguns dos métodos mais importantes utilizados na área.

2.1 Circuitos digitais

Circuitos eletrônicos digitais são formados pela conexão de várias portas lógicas, que por sua vez são blocos básicos responsáveis por implementar as operações booleanas básicas. De acordo com a maneira em que as portas lógicas básicas são conectadas, circuitos eletrônicos digitais com funções mais complexas podem ser criados; tais funções podem armazenar, transmitir e transformar sinais binários. Quando estes circuitos estão construídos com componentes físicos distintos, dá-se o nome de circuito discreto. Quando vários circuitos estão miniaturizados e agregados em um único componente físico, dá-se ao componente o nome de circuito integrado. A grande maioria dos circuitos digitais atuais são circuitos integrados: apenas a título de ilustração, processadores de uso geral como os processadores Intel Core[1] são circuitos digitais formados por enormes arranjos de portas lógicas, conectadas de forma implementar um circuito integrado capaz de executar funções úteis aos usuários.

Dependendo da forma como são construídos, os circuitos digitais podem ser divididos em dois grandes grupos:

- **Circuitos digitais combinatórios:** são circuitos digitais cuja(s) saída(s) depende(m) única e exclusivamente do conjunto de sinais aplicados nas suas entradas em um instante de tempo. A figura 2.1 mostra um circuito combinatório

[1] Marca registrada da Intel Corporation.

(ou combinacional) simples, capaz de efetuar a soma binária (S) de duas entradas de um bit cada (as entradas são A e B) e gerar um sinal de "*vai um*" (C) em nível lógico alto caso o resultado não possa ser armazenado em um único bit.

- **Circuitos digitais sequenciais:** são circuitos digitais cuja(s) saída(s) depende(m) do conjunto de sinais aplicados na suas entradas em um instante de tempo e também do histórico dos sinais aplicados nas suas entradas. Os circuitos digitais sequenciais possuem ao menos um elemento de memória, que é responsável por armazenar sinais que foram produzidos por combinações anteriores de sinais aplicados em sua(s) entrada(s). A figura 2.2 mostra um circuito que é capaz de contar – em quatro bits – o número de variações que a sua entrada sofreu desde um instante inicial qualquer: a cada transição de nível lógico 0 para o nível lógico 1 na entrada (CLK), o valor das saídas (Q_0 a Q_3) é decrementado. Os quatro *flip-flops* JK formam o elemento de memória do circuito[2].

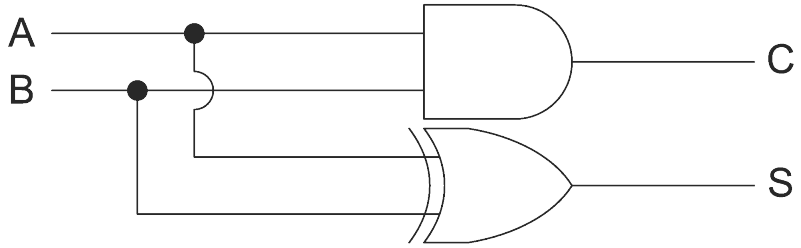


Figura 2.1. Circuito digital combinatório: somador de 1 bit

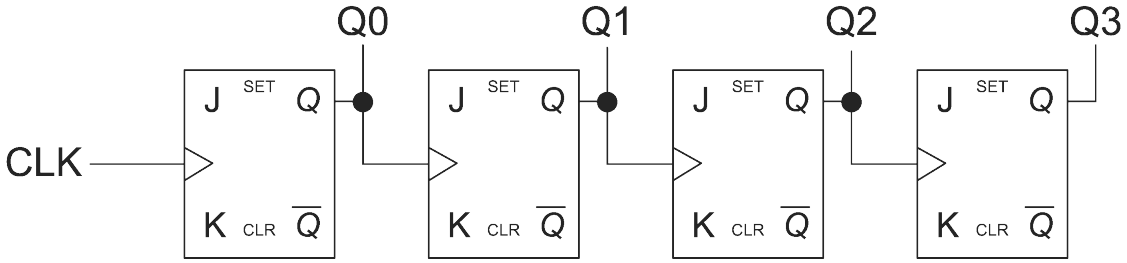


Figura 2.2. Circuito digital sequencial: contador assíncrono decrescente de 4 bits

O processo de desenvolvimento de um circuito digital integrado é dividido em várias etapas, mostradas na figura 2.3. Inicialmente, os projetos são especificados em uma linguagem de descrição de hardware, como Verilog ou VHDL [Smith, 1996]. A

[2]Apesar de não ser mostrado, todos os *flip-flops* JK mostrados no circuito da figura 2.2 devem possuir as entradas J e K em nível lógico alto e as entradas SET e CLR em nível lógico baixo para que o contador assíncrono assim implementado seja capaz de funcionar.

partir da especificação, o circuito é sintetizado e o resultado é uma série de portas lógicas conectadas. Neste ponto, é realizada a otimização de área, consumo de energia, desempenho ou testabilidade. As etapas seguintes são o mapeamento para uma mídia de implementação, onde o circuito pode ser descrito em nível de transistores seguida pela implementação física do mesmo [Wang et al., 2009].

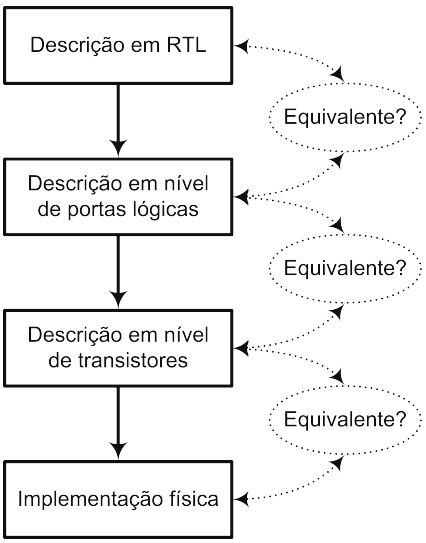


Figura 2.3. Etapas de desenvolvimento de um circuito integrado digital.

Durante todo o processo de desenvolvimento é preciso efetuar verificações entre as várias representações intermediárias do circuito, com o objetivo de minimizar a quantidade de falhas no produto final. A verificação pode ser feita comparando representações em níveis diferentes de abstração, comparando duas representações em mesmo nível ou ainda comparando uma representação em um nível qualquer com as especificações originais. Wang et al. [2009] mostra várias técnicas de verificação que podem ser utilizadas durante o processo de projeto de circuitos eletrônicos digitais:

1. **Simulação/emulação lógica e simulação do circuito**, que permitem verificar os requisitos de funcionalidade e temporização do projeto. Estas técnicas no entanto consomem muito tempo e podem efetuar uma busca incompleta por erros no projeto.

2. **Verificação funcional**, onde modelos que descrevem padrões de funcionalidade e comportamento são elaborados e então utilizados para verificar o comportamento funcional do projeto.

3. **Verificação formal**, que efetua a verificação contra modelos de referência chamados de "*golden models*". Os modelos de referência incluem uma lista de

propriedades funcionais/comportamentais que serão verificadas através de técnicas de verificações de modelos (do inglês *model checking*) e um circuito de referência, que será utilizado nas técnicas de verificação de equivalência.

De acordo com a técnica de verificação formal, dois circuitos digitais são ditos equivalentes quando o mesmo conjunto de sinais aplicados na entrada (ou a mesma sequência de sinais aplicados na entrada) gera o mesmo resultado na saída (ou a mesma sequência de resultados na saída). Molitor & Mohnke [2004] propõe uma definição formal para a equivalência de dois circuitos:

Definição 2.1.1. Dadas duas representações d_f e d_g de duas funções booleanas (com n e m entradas, respectivamente), $f, g : \{0,1\}^n \rightarrow \{0,1\}^m$, as duas funções booleanas f e g são equivalentes se $f(\alpha) = g(\alpha) \forall \alpha \in \{0,1\}^n$ e $n = m$.

É importante ressaltar que a verificação de equivalência de dois circuitos digitais só é possível se ambos pertencerem ao mesmo grupo: não é possível verificar equivalência entre um circuito digital combinatório e um circuito digital sequencial. O presente trabalho trata da verificação de equivalência de circuitos combinatórios.

2.2 Verificação de circuitos combinatórios

Vários métodos de verificação formal vêm sido desenvolvidas ao longo do tempo para lidar com o problema de verificação de equivalência. Silva & Glass [1999] separa essas técnicas em dois grupos, conforme pode ser visto na figura 2.4:

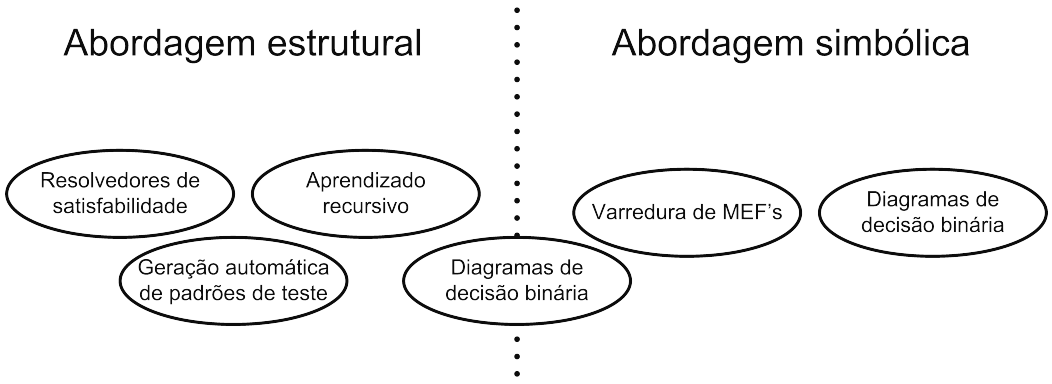


Figura 2.4. Métodos de verificação formal utilizados na verificação de circuitos.

A técnica de varredura de máquinas de estados finitos (MEF's) só é aplicável a verificação de circuitos digitais sequenciais, o que foge do escopo deste trabalho. As demais técnicas serão descritas em maiores detalhes nas sessões a seguir.

2.2.1 Diagramas de decisão binária

Diagramas de decisão binária (BDD - *Binary Decision Diagrams*) e diagramas de decisão binária reduzidos e ordenados (ROBDD - *Reduced Ordered Binary Decision Diagrams*) são estruturas de dados capazes de representar uma função booleana através de um um grafo direcionado acíclico, onde os nodos intermediários representam as variáveis da função em questão e os nodos folha representam os valores da função [Lee, 1959][Akers, 1978].

A criação de BDD's para uma função booleana qualquer se faz através de uma árvore de decisão binária; desde que cada variável apareça uma única vez no caminho da raiz até as folhas e que em qualquer caminho entre a raiz e as folhas a ordem em que as variáveis aparecem seja a mesma. Na figura 2.5 há um exemplo de diagrama de decisão binária em sua versão baseada em árvore de decisão para a função booleana $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c)$.

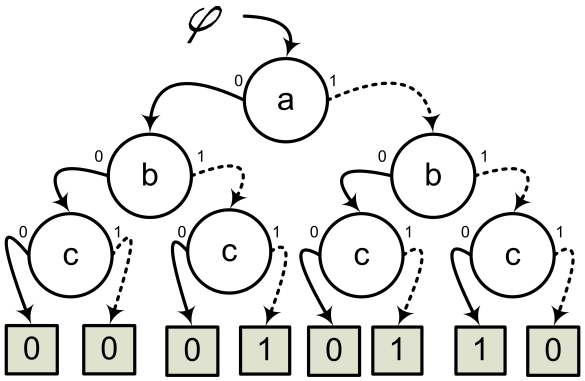


Figura 2.5. Um exemplo de diagrama de decisão binária para a função booleana $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c)$.

Na implementação, no entanto, todos os BDD's já são criados e manipulados em sua forma reduzida, tanto que diversos autores consideram BDD e ROBDD como termos sinônimos [Clarke et al., 1999]. Para compactar a representação de várias funções booleanas em um sistema baseado em ROBDD's, funções booleanas mais simples que têm seu ROBDD já representado em um ROBDD maior podem ser representadas como um novo ponto de entrada (simbolizado por uma seta) no ROBDD maior, como mostra a figura 2.6, onde existem duas funções representadas, φ e σ.

Um BDD pode ser transformado em um ROBDD aplicando as seguintes regras enquanto for possível:

1. Agregue todos os nodos folha com o mesmo rótulo em um único nodo, garantindo que exista apenas um nodo folha para cada rótulo de nodo folha.

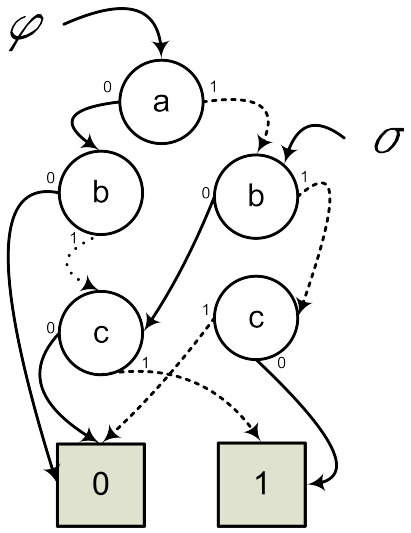


Figura 2.6. Um exemplo de diagrama de decisão binária reduzido e ordenado para a função booleana $\varphi = (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c) \vee (a \wedge \neg b \wedge c)$ e para a função booleana $\sigma = (\neg b \wedge c) \vee (b \wedge \neg c)$.

2. Faça a união de dois ou mais nodos duplicados (com os mesmos rótulos e com os mesmos filhos);

3. Se ambos os ponteiros de um nodo pai vão em direção a um mesmo nodo filho retire o nodo pai, pois ele é redundante.

A transformação de BDD's em ROBDD's é importante não apenas pela economia de espaço de armazenamento, mas também porque ROBDD's são estruturas de dados canônicas, isto é, dada uma ordem fixa de variáveis, existe um único ROBDD que representa aquela função booleana. Desta forma, para comparar duas funções booleanas quaisquer basta verificar se seus ROBDD's são iguais, operação de complexidade constante [Clarke et al., 1999].

Os primeiros trabalhos que propuseram utilizar ROBDD's na solução do problema de verificação de equivalência combinatória foram Fujita et al. [1988] e Malik et al. [1988]. Mais adiante, o trabalho de van Eijk & Janssen [1994] utiliza ROBDD's e explora similaridades[3] de circuitos sob verificação, conseguindo uma sensível melhora no tempo de execução do verificador. Algum tempo depois, Bollig & Wegener [1996] mostra que definir um ordem ótima de variáveis para que um ROBDD ocupe o menor espaço possível é um problema NP-difícil e Bryant [1998] mostra que, dependendo da ordem em que as as variáveis aparecem em um ROBDD (partindo da raiz em direção

[3]Um circuito com grande similaridade é um circuito onde podemos observar padrões de estruturas lógicas que se repetem várias vezes para formar o circuito como um todo. Em um circuito com baixa similaridade é difícil, ou impossível, encontrar padrões de estruturas lógicas que são repetidas.

às folhas), o número de vértices do diagrama pode variar entre $2n$ e $2^{n+1} - 2$ para uma função booleana qualquer com $2n$ variáveis. No trabalho de Bryant [1998] é também provado que o número de vértices de um ROBDD para verificar um multiplicador inteiro de tamanho n tem limite inferior $\Omega(1,09^n)$.

Mais recentemente, os trabalhos de Murgai et al. [1999], Scholl et al. [2001], Xu et al. [2003] e Mohammadi et al. [2009] propõe heurísticas interessantes para ordenação de variáveis, mas na prática a utilização de ROBDD's é limitada a circuitos mais simples. A chamada "explosão exponencial de estados" inviabiliza a utilização de ROBDD's na verificação de circuitos aritméticos contendo multiplicadores [Van Der Schoot & Ural, 1996] ou circuitos com baixa similaridade.

2.2.1.1 Diagramas de momento binário

Os diagramas de momento binário (BMD - *Binary Moment Diagrams*) e diagramas de momento binário multiplicativo (*BMD - *Multiplicative Binary Moment Diagrams*) foram inicialmente propostos por Bryant & Chen [1995] e são generalizações da idéia empregada em BDD's: ao contrário de trabalharem com representação de bits nos nodos folha, os BMD's trabalham com representação de palavras, levando a uma forte redução no espaço de armazenamento (se comparado com os BDD's e ROBDD's).

Técnicas que utilizam BMD's (ou variações, como *BMD's) são capazes de verificar multiplicadores inteiros [Wallace, 1964; Luk & Vuillemin, 1983] em tempo polinomial [Keim et al., 2003; Hamaguchi et al., 1995; Chen & Chen, 2001], mas não lidam de maneira satisfatória com funções lógicas booleanas [Becker et al., 1997]. Para contornar este problema, foram propostas extensões à estrutura original, buscando abordagens híbridas entre BDD's e BMD's. Clarke et al. [1995] definiu diagramas de decisão híbridos (HDD - *Hybrid Decision Diagrams*) e em um trabalho posterior [Clarke & Zhao, 1995] conseguiu verificar circuitos de divisão e de cálculo de raiz quadrada baseados no algoritmo de divisão de Sweeney-Robertson-Tocher. Drechsler et al. [1997] também utiliza uma estrutura de dados híbrida, o K*BMD - *Kronecker Multiplicative Binary Moment Diagram*, implementando uma aplicação que efetua substituições que é capaz de verificar circuitos aritméticos em tempo $O[log(n)]$ (verificado experimentalmente).

Apesar de bons resultados terem sido obtidos, encontrar boas ordenações de variáveis para BMD's e suas variantes ainda é um problema em aberto, pois, como nos ROBDD's, uma infeliz ordenação de variáveis pode levar à explosão de estados [Drechsler et al., 1997]. Além disso, verificadores baseados em BMD apenas conseguem provar que multiplicadores sem problemas estão realmente sem problemas em tempo razoável,

mas não há provas de que um verificador baseado em BMD consiga detectar falhas em um multiplicador com falhas em tempo razoável [Wefel & Molitor, 2000].

2.2.2 Geração automática padrões de teste

A verificação de equivalência através da geração automática de padrões de teste (ATPG - *Automatic Test Pattern Generation*) busca gerar automaticamente padrões de teste que sejam capazes de indicar problemas no circuito sob verificação, dado um circuito referência. É uma técnica bastante semelhante à técnica de verificação baseada em satisfabilidade (SAT), se diferenciando desta apenas no tipo de dados manipulados. Enquanto ATPG opera sobre redes booleanas, técnicas baseadas em SAT operam principalmente sobre fórmulas na forma normal conjuntiva (CNF - *Conjunctive Normal Form*). Além disso, técnicas de verificação que utilizam SAT surgiram originalmente no contexto de prova automática de teoremas, enquanto as técnicas de verificação baseadas em ATPG foram sugiram dentro da área de testes para circuitos [Biere & Kunz, 2002].

A idéia por trás da verificação por geração automática de padrão de testes é gerar um padrão capaz de gerar uma saída diferente em um circuito sob teste se comparada com a saída de um circuito de referência. A comparação do circuito sob teste com o circuito de referência é realizada através da criação de um *miter* [Brand, 1993; Goldberg & Novikov, 2002]. Um *miter* é um circuito criado conectando cada entrada comum do circuito sob teste e do circuito referência e conectando também a saída de cada circuito por meio de uma porta lógica com a função XOR[4], conforme pode ser visto na figura 2.7. Quando for encontrado um padrão para a entrada do *miter* que leve a saída para verdadeiro, o padrão é marcado como padrão gerador de falha.

A produção do padrão de teste é feita baseada em um modelo chamado de *stuck-at-fault*. Este modelo consiste no assinalamento de um sinal do circuito em um nível lógico constante (*stuck-at-0* ou *stuck-at-1*, para assinalamento em 0 ou em 1, respectivamente) e na busca de um valor para as entradas (chamado de vetor de teste) que ative a falha (ou seja, que levaria o sinal mantido constante ao nível contrário daquele previamente definido). Este processo de busca recebe o nome de justificativa e é aplicado a partir do ponto de assinalamento em direção às entradas do circuito. Caso a justificativa atribua a uma mesma variável dois valores lógicos diferentes é detectado um conflito e o assinalamento realizado não pode ser produzido.

[4]Quando o circuito possuir mais de uma saída, cada saída comum dos dois circuitos é conectada a uma XOR e a saída de cada XOR é conectada em uma porta OR com entradas suficientes.

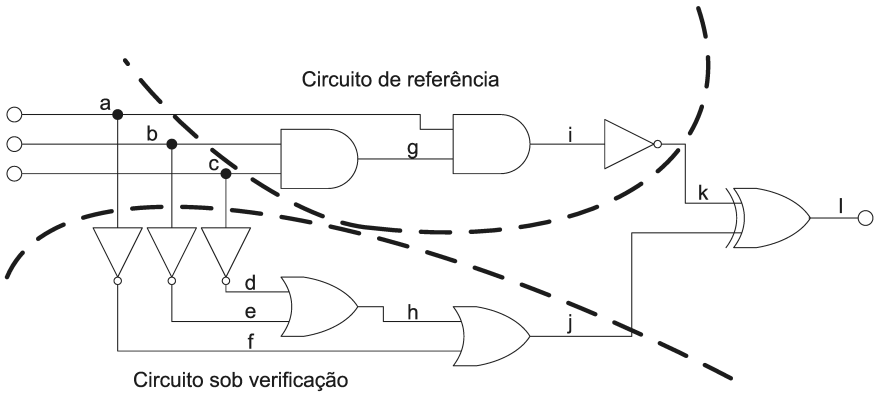


Figura 2.7. Exemplo de um *miter*.

Para justificar o valor de $l = 1$ na figura 2.8, é necessário que $j \neq k$; Seja a escolha de $j = 1$ e $k = 0$. Como $k = 0$, temos $i = 1$ e, por consequência, $a = 1$ e $g = 1$. $g = 1$ implica em $b = 1$ e $c = 1$. Na parte inferior do circuito, $a = 1$ leva a $f = 0$, $b = 1$ leva a $e = 0$ e $c = 1$ leva a $d = 0$. Como $d = 0, e = 0, f = 0$ tem-se $h = 0$ e por consequência, $j = 0$, que é um conflito, pois foi suposto $j = 1$. Uma vez encontrado o conflito, é realizado *backtracking* até a escolha feita e procura-se uma nova possibilidade que satisfaça $l = 1$.

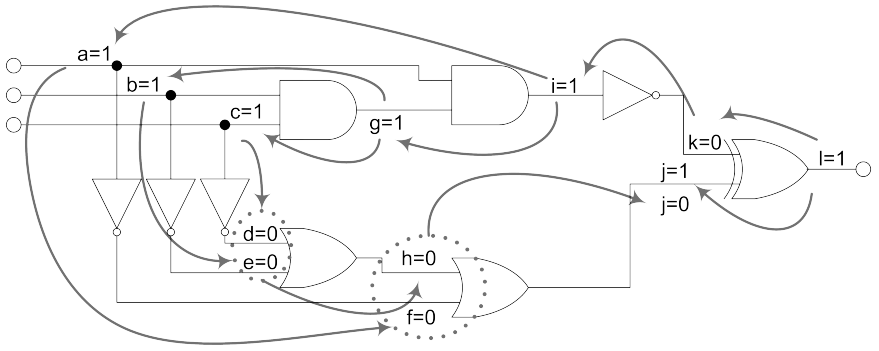


Figura 2.8. Fluxo do processo de justificativa de assinalamento de $l = 1$ resultando em conflito em j.

A nova possibilidade que satisfaz $l = 1$ é $j = 0$ e $k = 1$. A figura 2.9 mostra o processo de justificativa dessa nova situação: $k = 1$ leva a $i = 0$; $j = 0$ implica em $f = 0$ e $h = 0$, que por sua vez implica em $d = 0$ e $e = 0$. e, f e d iguais a zero implicam, respectivamente, em b, a e c iguais a 1; que por sua vez levam a $g = 1$ e $i = 1$, o que é um novo conflito. Como não podem ser feitas novas escolhas que satisfaçam $l = 1$, não existe um padrão nas entradas do *miter* que contradigam a saída *s-a-0* e os circuitos são equivalentes.

Mesmo sendo relativamente simples o processo de justificativa, a geração au-

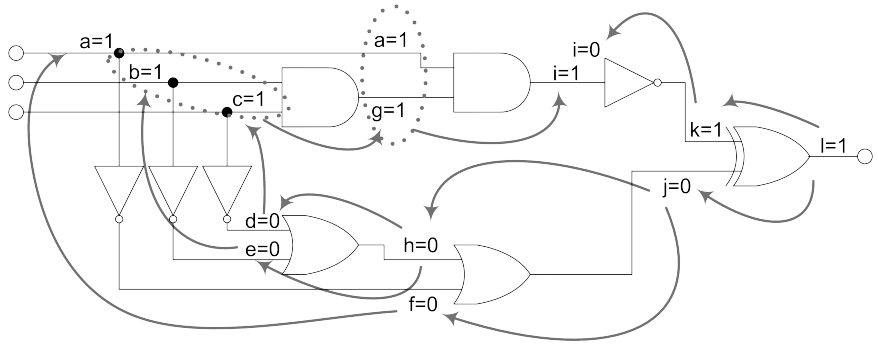


Figura 2.9. Fluxo do processo de justificativa de assinalamento de $l = 1$ resultando em conflito em i.

tomática de padrão de teste é um problema pertencente à classe de problemas NP-difícil, não havendo portanto um algoritmo que possa resolvê-lo em tempo polinomial [Ibarra & Sahni, 1975]. Com o objetivo de reduzir o tempo de execução, várias heurísticas foram utilizadas; a maioria tenta encontrar assinalamentos de sinais necessários para efetuar testes o mais cedo possível e/ou tenta realizar a busca de padrões no mínimo possível acima do espaço de decisão.

Roth [1966] mostrou um primeiro algoritmo completo para geração automática de padrões de testes, chamado de *Algoritmo-D*. Deste trabalho foram derivados novos algoritmos, como Fujiwara & Shimono [1983], Kirkland & Mercer [1987], Schulz et al. [1988], Bhattacharya & Hayes [1989], Lee & Ha [1990], Tafertshofer et al. [2000] e Gizdarski & Fujiwara [2001].

2.2.3 Aprendizado recursivo

O aprendizado recursivo (*recursive learning* - RL), técnica proposta por Kunz [1993] e depois melhorada em Kunz & Pradhan [1994] buscando tirar proveito da similaridade entre os circuitos verificados com isso melhorar a eficiência do ATPG [Schulz et al., 1988], além de permitir a execução de verificações em situações onde as técnicas de ATPG falham. A figura 2.10 mostra uma situação onde o método proposto por Schulz et al. [1988] não funciona quando $h = 0$; isso acontece porque os sinais e e x_4 não tem nenhuma relação entre si.

Essa técnica se baseia na análise de recursiva portas lógicas conectadas em um circuito combinatório, buscando extrair todas as dependências lógicas do circuito (a etapa de aprendizado) e a partir deste ponto elaborar implicações precisas a partir da atribuição de valores a determinados sinais booleanos. Elaborar implicações precisas significa identificar todos os valores de sinais que são definidos em consequência de uma atribuição já realizada.

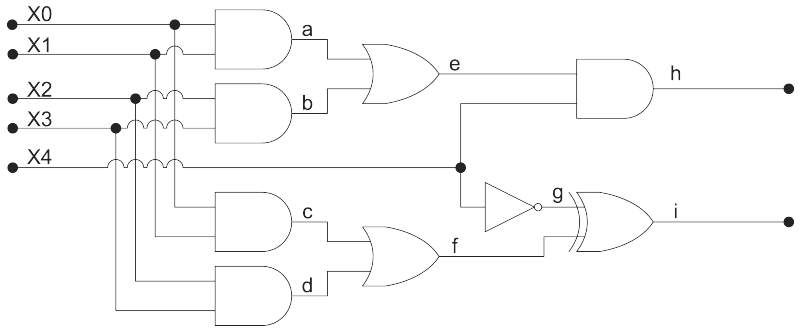


Figura 2.10. Circuito para exemplificar o processo de aprendizado recursivo.

O processo de aprendizado é dito recursivo porque é iniciado em uma porta lógica em um determinado circuito e, uma vez que determinados valores de entrada ou saída não podem ser definidos, o processo é repetido recursivamente para a porta lógica que está conectada naquela entrada ou saída. Seja o circuito mostrado na figura 2.10 e as definições a seguir:

Definição 2.2.1. Portas lógicas justificadas ou não justificadas: seja G uma porta lógica em uma rede combinatória que tem pelo menos um sinal de entrada ou saída especificado. G é dita não justificada se existem um ou mais sinais na(s) entrada(s) ou saída(s) de G para os quais é possível encontrar um valor que leve a um conflito entre os sinais daquela porta lógica. Caso tais sinais não existam, G é dita justificada.

Definição 2.2.2. Justificativa: sejam $f_0, f_1, f_2, \ldots, f_n$ uma série de sinais de entrada ou saída não especificados de uma porta lógica G não justificada. Seja $J = \{f_0 = V_0, f_1 = V_1, f_2 = V_2, \ldots, f_n = V_n\}$ um conjunto de assinalamentos para $f_0, f_1, f_2, \ldots, f_n$. J é chamado de justificativa de G se as combinações tornam G uma porta lógica justificada.

Definição 2.2.3. Conjunto completo de justificações: seja G_C um conjunto de justificações $J_1, J_2, J_3, \ldots, J_n$ para uma porta lógica G não justificada. Se existir pelo menos uma justificação $J_i \in G_C, 1 \leq i \leq n$ para qualquer justificação possível J^* de G de forma que $J_i \subseteq J^*$, então G_C é chamado de conjunto completo de justificações.

Definição 2.2.4. Implicações simples: implicações simples são implicações na forma $a \rightarrow b$ em que a é um assinalamento escolhido em uma porta lógica e b é um assinalamento obtido através da propagação do sinal a através daquela porta lógica.

Definição 2.2.5. Implicações diretas: implicações diretas são implicações na forma $a \rightarrow b$ em que a é um assinalamento escolhido em uma porta lógica e b é um assinalamento obtido através da propagação do sinal a de mais de uma porta lógica.

Definição 2.2.6. Implicações indiretas: implicações indiretas são implicações na forma $a \rightarrow b$ em que a é um assinalamento na saída de uma porta lógica não justificada e b é qualquer assinalamento obtido através da intercessão de todos assinalamentos obtidos pelas implicações simples e diretas para cada justificação possível de a. Em outras palavras, as implicações indiretas correspondem às implicações obtidas pela intercessão dos conjuntos que fazem parte do conjunto completo de justificações.

Supondo $h = 0$, então tem-se $e = 0$ ou $x_4 = 0$ (ou seja, $\overline{h} \rightarrow \overline{e}$ e $\overline{h} \rightarrow \overline{x_4}$, que são implicações simples). Como a porta com saída h não está justificada devido a indefinições nas suas entradas, é preciso supor um valor para alguma das suas entradas e aplicar a técnica de aprendizado nas portas lógicas que controlam aquelas entradas. Supondo $e = 0$, então $a = 0$ e $b = 0$, o que justifica a porta OR. Como $a = 0$, x_0 ou x_1 valem 0 e então tem-se $c = 0$. Por analogia, se $b = 0$ então $d = 0$. Sendo $c = 0$ e $d = 0$ tem-se $f = 0$ ($\overline{e} \rightarrow \overline{f}$, uma implicação indireta). Observe que, devido à grande similaridade estrutural do circuito, os valores c, d e f puderam ser facilmente determinados (calculando implicações indiretas), sem no entanto obter diretamente valores para as entradas. Para determinar um conjunto de valores que justificam $i = 0$ nesta situação basta definir $g = 0$ (já que $f = 0$), o que ainda justifica totalmente a porta com saída $h = 0$, que agora tem a entrada $e = 0$ e a outra entrada com valor 1.

Em circuitos com grande similaridade como aquele da figura 2.10 o aprendizado recursivo pode trazer grandes vantagens já que valores "aprendidos" podem ser reutilizados, o que acaba por diminuir a profundidade de outras pesquisas. Esta abordagem, no entanto, é intratável para circuitos com baixa similaridade: o tempo de execução cresce exponencialmente com o nível de recursão. Apesar disso, nos casos de alta similaridade, bons resultados foram obtidos [Silva & Silva, 1999].

2.2.4 Verificação baseada em resolvedores SAT

O problema de satisfabilidade booleana foi um primeiros problemas pertencentes à classe de problemas NP-completo [Cook, 1971] e pode ser definido como se segue:

Definição 2.2.7. Dada uma fórmula proposicional γ, o problema de satisfabilidade booleana em γ se resume em determinar se existe um conjunto de atribuições de variáveis de γ que faz com que a expressão seja verdadeira. Se existir tal atribuição, a fórmula é chamada de satisfazível, caso contrário é chamada de insatisfazível.

Fórmulas proposicionais podem ser escritas sob várias formas, mas a maioria dos trabalhos relacionados a SAT lidam com a forma normal conjuntiva (CNF - *Conjunctive Normal Form*). A forma normal conjuntiva é um padrão de notação para fórmulas proposicionais com a seguinte definição:

Definição 2.2.8. Uma fórmula proposicional está na forma normal conjuntiva se estiver escrita como uma conjunção de cláusulas formadas pela disjunção de literais, que podem estar apresentados na sua forma positiva ou negativa.

A equação 2.1 mostra uma fórmula proposicional γ composta de 3 cláusulas, cada uma delas formada pela disjunção de literais σ_1, σ_2 e σ_3 em sua forma positiva (σ) ou negativa ($\neg\sigma$).

$$\gamma = \overbrace{\underbrace{(\neg\sigma_1 \vee \sigma_2 \vee \sigma_3)}_{\text{cláusula}} \wedge \underbrace{(\sigma_1 \vee \sigma_2 \vee \neg\sigma_3)}_{\text{cláusula}} \wedge \underbrace{(\sigma_1 \vee \neg\sigma_2 \vee \sigma_3)}_{\text{cláusula}}}^{\text{forma normal conjuntiva}} \tag{2.1}$$

Além de ser alvo de várias pesquisas na área de verificação de equivalência combinatória [Larrabee, 1992; Goldberg et al., 2001; Moskewicz et al., 2001; Goldberg & Novikov, 2002; Eén & Sörensson, 2003; Prasad et al., 2005; Oliveira, 2006; Andrade et al., 2007; Mendes, 2008], o problema da satisfabilidade booleana tem aplicações em pesquisa operacional [Barth & Stadtwald, 1995], na área de inteligência artificial [Bayardo & Schrag, 1997; Zhang, 1997], na resolução de problemas de BCP (*binate covering problem*) [Coudert, 1996; Ferrandi et al., 1998; Hatchel & Somenzi, 2000; Flores et al., 2001] entre outras áreas da computação.

Apesar de ter aplicabilidade em várias áreas, as pesquisas envolvendo resolvedores SAT são relativamente recentes. O motivo para isto é que apenas à época do trabalho de Larrabee [1990] as implementações começaram a apresentar eficiência de forma a apresentarem uma solução tratável para problemas nas áreas aqui mencionadas. O trabalho de Larrabee [1992] foi um dos primeiros a propor a utilização de resolvedores SAT em problemas de ATPG com circuitos combinatórios. Mais tarde, Silva & Sakallah [1996] apresentou o GRASP (*Generic seaRch Algorithm for the Satisfiability Problem*); um resolvedor SAT com aprendizado de cláusulas de conflito e retrocesso não cronológico que foi capaz de resolver em alguns minutos problemas até então considerados intratáveis. Depois de GRASP apareceram vários outros trabalhos propondo melhorias ne eficiência dos resolvedores SAT, sendo que a maioria deles são baseados no algoritmo proposto por Davis, Putnam, Logemann e Loveland (conhecido como resolvedor SAT DPLL) [Davis et al., 1962], uma melhoria do trabalho de Davis & Putnam [1960].

Os primeiros trabalhos a utilizarem resolvedores SAT com aplicação explícita em verificação de equivalência combinatória foram Gupta & Ashar [1998] e Silva & Silva [1999]. Enquanto o primeiro mesclou resolvedores SAT com técnicas de BDD o segundo integrou aprendizado recursivo e um resolvedor SAT. Algum tempo depois, Goldberg et al. [2001] utilizou exclusivamente técnicas de resolvedores SAT para obter resultados semelhantes àqueles obtidos com técnicas de ATPG e BDD's/BMD's; feito que ainda não havia sido atingido e que efetivamente demonstrou que resolvedores SAT são uma alternativa viável para resolver o problema de verificação de equivalência combinatória.

2.2.4.1 Estrutura de um resolvedor SAT

Conforme já mencionado, a maioria dos resolvedores SAT atuais são baseados no resolvedor proposto por Davis et al. [1962], um refinamento do trablho de Davis & Putnam [1960], que por sua vez foi elaborado para verificar fórmulas lógicas de primeira ordem.

Resolvedores modernos, como o zChaff [Moskewicz et al., 2001], o BerkMin [Goldberg & Novikov, 2002], e o MiniSat [Eén & Sörensson, 2003] usam o algoritmo DPLL [Davis et al., 1962] com retrocesso não-cronológico por análise e aprendizado de cláusulas de conflito [Silva & Sakallah, 1996] e propagação de restrições booleanas com literais vigiados.

Esses resolvedores SAT dirigidos por conflito (como são chamados os resolvedores os citados anteriormente) possuem uma estrutura de dados capaz de representar cláusulas e assinalamentos, utilizam um mecanismo de inferência conhecido como propagação unitária (ou unária) e realizam uma busca a fim de de derivar informações a partir de uma fórmula CNF.

A propagação unitária é um algoritmo em que, tão logo uma cláusula CNF torne-se unitária (apenas um termo verdadeiro) devido ao assinalamento corrente, o literal resultante é assinalado como verdadeiro e como consequência, a variável é assinalada como falsa ou verdadeira (dependendo dela estar complementada ou não naquela cláusula) e então avaliado seu resultado em outras cláusulas das quais aquela variável faz parte. Este processo é executado seguidamente até que nenhuma informação possa mais ser propagada. Depois disto, através de uma heurística, variáveis são selecionadas e tem valores assinalados até que haja um conflito, que ocorre quando todos os literais de uma cláusula foram assinalados como falsos, o que leva à expressão CNF ser falsa (o que é um conflito pois, por construção, a saída do *miter* representado pela fórmula CNF deve ser verdadeira). Neste momento, uma cláusula de conflito é construída e adicionada ao banco de cláusulas. Os assinalamentos efetuados são então anulados por

uma técnica de retrocesso até que a cláusula de conflito torne-se unitária e possa ser propagada para a continuação do procedimento. O algoritmo 1 mostra com a estrutura básica de resolvedor SAT moderno.

Algoritmo 1: Algoritmo DPLL, no qual a maioria dos resolvedores SAT são baseados

```
Entrada: expressão booleana escrita na forma normal conjuntiva (CNF).
Saída: SAT caso a expressão seja satisfazível, UNSAT caso contrário.
while Verdadeiro do
  Propaga();
  if não existe conflito then
    if Todas variáveis já foram assinaladas then
      Retorne SAT;
    else
      Decide SAT;
    end if
  else
    Analisa();
    if Foi encontrado conflito na raíz then
      Retorne UNSAT;
    else
      Retrocede();
    end if
  end if
end while
```

2.2.4.2 Técnicas SAT baseadas em pré-processamento

Alguns trabalhos que envolvem técnicas SAT para executar verificação de equivalência combinatória não trabalham em evoluções nos resolvedores SAT em si, mas aplicando algum tipo de processamento nas informações que são repassadas ao resolvedor. Estas técnicas se dividem em duas categorias:

- Técnicas que diminuem o número de variáveis e/ou cláusulas do problema original: se baseia na idéia de que, quanto menor a instância do problema de SAT, menor o tempo de verificação.
- Técnicas que aumentam o número de cláusulas do problema original; adicionando cláusulas redundantes de forma a melhorar a eficiência do procedimento de busca do resolvedor SAT.

Em Silva [2000], simplificações algébricas e regras de inferência para cláusulas unitárias e binárias são utilizadas para manipular o problema original com o objetivo

de melhorar o desempenho do resolvedor SAT. Bacchus & Winter [2003] também efetua simplificações algébricas baseadas em técnicas de hiper-resolução binária para modificar as cláusulas CNF de um circuito a ser verificado, sendo capaz de, inclusive, detectar conflitos antes mesmo do resolvedor SAT ser executado[5].

Em Subbarayan & Pradhan [2004], um preprocessador de instâncias SAT - NiVER - foi apresentado, sendo capaz de efetuar o preprocessamento de problemas verificação sem aumentar o número de cláusulas CNF a serem tratadas pelo resolvedor SAT, conseguindo significativas melhorias no tempo de execução do BerkMin [Goldberg & Novikov, 2002], um resolvedor SAT considerado estado da arte. Arora & Hsiao [2003] e Arora & Hsiao [2004] utilizam a produção de implicações diretas, indiretas e estendidas a partir de um circuito para aumentar a base de pesquisa do resolvedor, aumentando em quase mil vezes o desempenho do processo de verificação.

Andrade et al. [2008b] também utiliza implicações propostas por Kunz & Pradhan [1993] e Zhao et al. [1997] conseguindo melhorar o tempo de execução do BerkMin [Goldberg & Novikov, 2002]; em alguns casos, inclusive, melhorando os resultados obtidos com o NiVER [Subbarayan & Pradhan, 2004]. Este trabalho é mostrado em maiores detalhes na seção 2.3.2.

2.3 Trabalhos relacionados

Esta seção mostra detalhes dos trabalhos mais importantes relacionados ao tema desta dissertação. Inicialmente é abordada a ferramenta *Hypre*, proposta por Bacchus & Winter [2003], sendo seguida pela abordagem da ferramenta *Vimplic*, de Andrade et al. [2008b].

2.3.1 Pré-processamento de cláusulas CNF utilizando hiper-resolução binária

A hiper-resolução uma estratégia de resolução proposta por Robinson [1965] que efetua resoluções lógicas envolvendo mais de duas expressões lógicas. O resultado de um passo de hiper-resolução equivale ao resultado de se aplicar sequencialmente uma série de passos de resoluções binárias comuns (que envolvem apenas duas expressões), com a diferença que no caso da hiper-resolução, resultados intermediários não são gerados.

Ao contrário do que acontece com cláusulas de conflito, a adição de eventuais cláusulas de passos de resolução intermediária é contraprodutiva para resolvedores SAT,

[5]Este trabalho é mostrado em maiores detalhes na seção 2.3.1

pois apenas aumentam o tamanho da instância a ser resolvida [Bacchus & Winter, 2003]. Dentro deste contexto, executar hiper-resolução como pré-processamento de cláusulas CNF em uma instância SAT pode ajudar a reduzir o tamanho do problema original sem gerar cláusulas adicionais, tendo como consequência imediata a redução do tempo de execução do resolvedor. No caso específico de problemas de verificação de equivalência combinatória, Bacchus & Winter [2003] propõe a utilização de uma regra de hiper-resolução, chamada de hiper-resolução binária e que é mostrada a seguir:

Definição 2.3.1. Regra de hiper-resolução binária: A regra de hiper-resolução binária (*HyperBinRes*) é definida por como:

$$\frac{(l_1, l_2, l_3, \ldots, l_n)(\overline{l_1}, l)(\overline{l_2}, l)(\overline{l_3}, l)\ldots(\overline{l_{n-1}}, l)}{(l, l_n)}, \text{ para } n \geq 2 \tag{2.2}$$

Seja uma fórmula CNF:

$$\gamma = (\sigma_1 \vee \sigma_2 \vee \sigma_3 \vee \sigma_4) \wedge (\sigma_5 \vee \overline{\sigma_2}) \wedge (\sigma_5 \vee \overline{\sigma_1}) \wedge (\sigma_5 \vee \overline{\sigma_4}) \tag{2.3}$$

A aplicação de de hiper-resolução binária em γ gera uma única cláusula $(\sigma_3 \vee \sigma_5)$. Isto é equivalente a aplicar sucessivamente os seguintes passos de resolução binária:

$$\begin{aligned}(\sigma_1 \vee \sigma_2 \vee \sigma_3 \vee \sigma_4) \wedge (\sigma_5 \vee \overline{\sigma_2}) \wedge (\sigma_5 \vee \overline{\sigma_1}) \wedge (\sigma_5 \vee \overline{\sigma_4}) &= \\ (\sigma_1 \vee \sigma_3 \vee \sigma_4 \vee \sigma_5) \wedge (\sigma_5 \vee \overline{\sigma_1}) \wedge (\sigma_5 \vee \overline{\sigma_4}) &= \\ (\sigma_3 \vee \sigma_4 \vee \sigma_5) \wedge (\sigma_5 \vee \overline{\sigma_4}) &= \\ (\sigma_3 \vee \sigma_5)\end{aligned}$$

Ao reduzir cláusulas com n literais, a hiper-resolução binária pode produzir cláusulas unitárias: seja a mesma expressão γ mostrada na equação 2.3, porém adicionada da cláusula $(\sigma_3 \vee \overline{\sigma_5})$. Aplicar hiper-resolução binária nesta nova expressão levaria a um cláusula unitária σ_3. Conforme será visto adiante, cláusulas unitárias tem grande potencial de redução na expressão CNF.

Um outro tipo de redução que pode ser realizada em cláusulas CNF é a regra da redução de igualdade, definida a seguir:

Definição 2.3.2. Regra da redução de igualdade: Seja uma fórmula booleana γ escrita em CNF que contenha tanto cláusulas $(\overline{a} \vee b)$ quanto cláusulas $(a \vee \overline{b})$, ou seja cláusulas $a \Rightarrow b$ e $b \Rightarrow a$. A redução de igualdade envolve substituir todas as ocorrências de b por a, seguida da remoção das cláusulas récem-tranformadas em tautologias $(a \vee \overline{a})$

seguida da simplificação de literais iguais duplicados em cada cláusula, ou seja $(a \vee a) = a$.

A redução de igualdade também pode reduzir o tamanho de cláusulas e produzir cláusulas unitárias. A ocorrência de cláusulas unitárias em expressões CNF permite fortes reduções no tamanho da expressão, de acordo com a definição a seguir:

Definição 2.3.3. Redução da expressão CNF por cláusulas unitárias: Dada uma fórmula booleana γ escrita em CNF com um conjunto cláusulas $\kappa_1 \wedge \kappa_2 \wedge \kappa_3 \wedge \ldots \wedge \kappa_n$. Seja também uma cláusula $\kappa_x \in \gamma$, ou seja, com $1 \leq x \leq n$. Se κ_x é formada apenas por um único literal σ, então γ pode ser transformada em uma expressão equivalente γ' retirando a cláusula κ_x e todas as ocorrências de $\overline{\sigma}$ nas cláusulas restantes.

A aplicação da regra da redução por cláusulas unitárias pode gerar novas cláusulas unitárias da expressão CNF. A aplicação sucessiva da regra da redução por cláusulas unitárias até que não possa mais ser aplicada é chamada de propagação unitária (UP, do inglês *unit propagation*) ou propagação da restrição booleana (BCP, do inglês *boolean constraint propagation*), a mesma utilizada na função **Propaga** presente no algoritmo 1 descrito na seção 2.2.4.

A idéia básica da ferramenta *Hypre* é aplicar a propagação unitária, a regra de redução de igualdade e a hiper-resolução binária até que nenhuma nova inferência possa ser aplicada por nenhuma dessas regras e a partir daí, alimentar a expressão simplificada a um resolvedor SAT. *Hypre* modela as cláusula da expressão CNF em um conjunto de estruturas de dados similar àquela mostrada pelo grafo de implicações total citado por Andrade [2008] e aplica então o algoritmo 2.

Algoritmo 2: Algoritmo de pré-processamento do Hypre [Bacchus & Winter, 2003].

```
Efetue a propagação de todas as cláusulas unitárias.
Encontre todos os componentes fortemente conectados.
Execute todas as reduções de igualdade.
Desmarque todos os nodos no grafo de implicações.
while Existir nodos desmarcados do
  for Cada estado ι que está marcado e não tem nodo pai do
    Visite(ι); /* Algoritmo 3 */
    Execute a busca de componentes fortemente conectados incremental;
    Desmarque os nodos de acordo com os dois casos acima;
  end for
end while
```

Algoritmo 3: Algoritmo de visitação do Hypre [Bacchus & Winter, 2003].

Entrada: Nodo a ser visitado (ι)
if ι está marcado **then**
  Retorne;
end if
$ImplicantesAtuais \leftarrow \{\}$
for Cada estado l tal que $(\bar{\iota}, l)$ que está no grafo de implicação **do**
  if $l \in ImplicantesAtuais$ **then**
    Remova $(\bar{\iota}, l)$ do grafo de implicação
  else
    Visite(l);
    if ι está marcado **then**
      Retorne;
    end if
    $ImplicantesAtuais \leftarrow ImplicantesAtuais \cup Descendentes(l)$
  end if
end for
$ImplicantesPU \leftarrow \{l$ tal que $UP(\iota) \vdash l\}$
if Contradição encontrada **then**
  $UP(\bar{\iota})$
  Marque todos os literais cujo valor booleano está definido
  Desmarque os nodos de acordo com os dois casos acima;
end if
$NovosImplicantes \leftarrow ImplicantesPU - ImplicantesAtuais$
for Cada $l \in NovosImplicantes$ **do**
  if $l \in NovosImplicantes$ **then**
    Continue;
  else
    Adicione $(\bar{\iota}, l)$ ao grafo de implicações
    Visite(l);
    $ImplicantesAtuais \leftarrow ImplicantesAtuais \cup Descendentes(l)$
  end if
end for
Marque ι

Inicialmente a ferramenta *Hypre* faz a leitura das cláusulas CNF no formato DIMACS[6] e executa a propagação de cláusulas unitárias. Terminado este passo, o algoritmo de Tarjan [1972] é executado e todos os componentes fortemente conectados são detectados e simplificados, deixando ativos apenas os literais de menor módulo que faziam parte dos componentes ativados; os demais literais do componente, agora desativados, têm sua aresta de entrada retirada (a não ser quando seus antecedentes

[6]O formato DIMACS é descrito no apêndice B.

também estão desativados, fazendo parte de um caminho totalmente desativado). Uma vez que novas cláusulas unitárias podem ser formadas, o algoritmo BCP é novamente executado e o processo de hiper-resolução é ativado.

Caso hajam redução de cláusulas (e consequente alteração no grafo de implicações), mais uma vez é executado BCP e nova busca de componentes fortemente conectados, novo BCP e nova etapa de hiper-resolução. Este ciclo é interrompido apenas quando a aplicação dos processos de BCP, do algoritmo para busca de componentes fortemente conectados e a hiper-resolução deixam de produzir alterações no grafo de implicações. Neste ponto são impressas as cláusulas CNF em um arquivo DIMACS, que pode ser passado para o resolvedor SAT. É importante dizer que o processo de eliminação de componentes fortemente conectados elimina quaisquer tautologias que possam surgir; no caso de contradições serem encontradas, o processo é interrompido indicando o fato e dispensando a execução do resolvedor SAT.

2.3.2 Derivação de implicações a partir de circuitos combinatórios

A ferramenta *Vimplic*, proposta por Andrade et al. [2008b], deduz implicações a partir de circuitos combinatórios e acrescenta tais informações ao problema original para então fornecer o problema modificado (mas equivalente ao original) para verificação no resolvedor SAT. Para execução de verificação de equivalência combinatória utilizando esta ferramenta, os seguintes passos devem ser executados em ordem:

1. Criação de um circuito *miter* para comparação entre o circuito sob verificação e um circuito de referência (passo comum a todos os processos de verificação de equivalência envolvendo resolvedores SAT);

2. execução da ferramenta *Vimplic* propriamente dita, para derivação de novas implicações e/ou adição de informações redundantes a partir do circuito *miter*;

3. conversão do circuito *miter* em uma expressão CNF e adição das informações adicionais geradas pela ferramenta *Vimplic*;

4. submissão da expressão CNF com as cláusulas extras a um resolvedor SAT.

As etapas 1 e 3 mostradas anteriormente são relativamente simples e estão descritas com maiores detalhes em Andrade [2008]. A etapa 4 consiste na aplicação dos resultados em um resolvedor SAT, como os propostos por Goldberg & Novikov [2002] ou Eén & Sörensson [2003]. A seção 2.3.2.1 descreve resumidamente o funcionamento

do *Vimplic*, sendo que a descrição mais detalhada pode ser encontrada em Andrade [2008]. Antes, no entanto, serão mostrados alguns conceitos importantes trabalhados por aquele autor.

Definição 2.3.4. Grafo de implicações: Um grafo dirigido $G(V, V, A)$, onde V é um conjunto de vértices formado por $V_1 \bigcup V_2$ e A um conjunto de arestas é chamado de grafo de implicações se: cada vértice $v \in V_1$ representa um sinal de um circuito (chamado de **vértice de sinal**) e é um vértice do tipo AND $v \in V_2$, que representa a conjunção de duas ou mais variáveis; cada aresta $a \in A$ representa uma implicação entre o vértice fonte e o vértice destino.

Definição 2.3.5. Grafo de implicações parcial: Um grafo de implicações parcial é um grafo de implicações que não possui vértices de convolução AND.

Um vértice em um grafo de implicações pode ser *ativado* ou não. Um vértice de sinal é chamado de *ativado* quando algum dos seus predecessores também está ativado ou quando o próprio é escolhido como ativado. Um vértice do tipo AND está ativado apenas quando todos os seus predecessores estão ativados ou quando o próprio foi escolhido como ativado.

O processo de construção de um grafo de implicações para uma porta lógica é bastante simples. Para cada uma de suas portas lógicas, basta assinalar uma de suas entradas para um valor e calcular derivações simples a partir daquele assinalamento. É necessário efetuar o assinalamento de cada um dos sinais da porta e calcular as derivações simples para obter o grafo de implicações daquela porta.

2.3.2.1 Funcionamento da ferramenta *Vimplic*

Inicialmente, a ferramenta lê a descrição do *miter* no formato BENCH (descrito no apêndice A) e constrói um grafo de implicações (de acordo com a definição 2.3.4) e uma tabela que armazena um assinalamento da saída de uma porta lógica que a torna não justificada (conforme a definição 2.2.1) e o conjunto completo de justificações que levam à porta ser justificada. Ambas estruturas são inicializadas vazias.

A segunda etapa é a adição de vértices e arestas no grafo de implicações inicialmente vazio e o preenchimento dos campos da tabela, procedimentos feitos à medida que as portas lógicas do *miter* forem encontradas. A terceira etapa consiste na varredura da estrutura topológica do circuito *miter* e, a cada porta lógica não justificável encontrada, a informação da tabela é recuperada e utilizada para derivação de implicações indiretas (ver definição 2.2.6).

A quinta e sexta etapas derivam implicações diretas (ver definição 2.2.5) e simples (ver definição 2.2.4) para o circuito encontrado. Ao final do processo, as implicações derivadas são convertidas para cláusulas CNF e um arquivo DIMACS (descrito no apêndice A) é salvo para ser adicionado ao circuito original convertido em CNF. O algoritmo executado por *Vimplic* é mostrado no algoritmo 4.

Algoritmo 4: Algoritmo Vimplic apresentado por Andrade [2008].

```
Leia o circuito miter no formato BENCH;
Crie um grafo de implicações vazio;
Crie uma tabela relacionando assinalamento/conjunto completo de justificações
vazia;
while Não processou todas as portas lidas do
  Leia uma porta lógica;
  Crie a representação a porta no grafo de implicações;
  if Porta é não justificável then
    Armazene o assinalamento da saída da porta e conjunto completo de
    justificações na tabela;
  end if
end while
Execute a busca em largura no circuito miter;
for Cada saída de porta lógica encontrada do
  a ← assinalamento da saída da porta
  C ← conjunto completo de justificações na tabela;
  DeriveImplicacoesIndiretas(a, C); /* Algoritmo 5 */
end for
```

Algoritmo 5: Algoritmo para derivação de implicações indiretas apresentado por Andrade [2008].

```
DerivaImplicacoesIndiretas(a, C)
Entradas: Assinalamento a; Conjunto completo de justificações C;
for Cada assinalamento b de C do
   DerivaImplicaçõesDiretas(b); /* Algoritmo 6 */
  Armazena assinalamentos obtidos;
end for
I ← interseção de todos os conjuntos de implicações diretas encontrados;
for Cada assinalamento i presente em I do
  Imprimir implicação b → i
end for
```

Algoritmo 6: Algoritmo para derivação de implicações diretas apresentado por Andrade [2008].

DerivaImplicacoesDiretas(b)
Entrada: Assinalamento b;
while Lista de assinalamentos não está vazia **do**
  DerivaImplicaçõesSimples(b); /* Algoritmo 7 */
  Adiciona o novo assinalamento à lista de assinalamentos;
end while

Algoritmo 7: Algoritmo para derivação de implicações simples apresentado por Andrade [2008].

DerivaImplicacoesSimples(b)
Entrada: Assinalamento b;
Executa busca em largura no grafo de implicações a partir do nó inicial b;
Armazena a lista de nós alcançados;

Capítulo 3

Desenvolvimento do trabalho

Este capítulo mostra em detalhes o funcionamento da ferramenta implementada a partir dos trabalhos de Andrade et al. [2008b] e Bacchus & Winter [2003]. A primeira seção mostra detalhes de implementação da ferramenta o funcionamento detalhado da ferramenta implementada neste trabalho e na seção seguinte são mostrados exemplos simples de funcionamento da ferramenta em situações de satisfabilidade e insatisfabilidade, com o objetivo de auxiliar a compreensão do funcionamento da mesma.

3.1 Implementação da ferramenta

Esta seção é dedicada a detalhar a implementação da ferramenta construída neste trabalho. De forma geral, a ferramenta carrega a descrição de um circuito *miter*, cria um grafo de implicações a partir do circuito, acrescenta implicações conforme recomendado por Andrade et al. [2008b] e então executa o processo de hiper-resolução da ferramenta *Hypre* sobre o grafo de implicações montado a partir do circuito. Mais detalhes podem ser vistos a seguir.

3.1.1 Leitura de um circuito combinatório e produção de implicações

O programa se inicia fazendo a leitura de um arquivo contendo a descrição de um circuito digital combinatório escrito no formato BENCH (descrito no apêndice A) e convertendo a entrada em um grafo que representa aquele circuito: no grafo, os nodos são as portas lógicas, arestas comuos sinais. Com exceção das portas XOR/XNOR, que só são permitidas em versões de duas entradas; os *buffers* e portas NOT, que só possuem uma entrada, as portas lógicas com mais de duas entradas são mapeadas no grafo como

se fossem um arranjo linear de várias portas lógicas de duas entradas. Assim, para cada porta lógica OR, NOR, AND ou NAND com $n > 2$ entradas, são gerados $n - 1$ nodos no grafo que representa o circuito e $n - 2$ sinais extras são adicionados. A figura 3.1 mostra a idéia empregada neste mapeamento, que é utilizado apenas para produção das implicações diretas e indiretas. Nas etapas de hiper-resolução e simplificação as portas OR, NOR, AND e NAND com múltiplas entradas são trabalhadas da maneira como são.

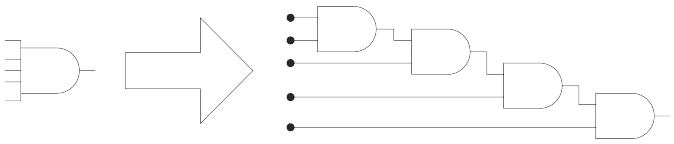


Figura 3.1. Mapeamento de portas lógicas com várias entradas realizado para criação do criação do grafo que representa o circuito.

Uma vez criado um grafo com a representação topológica do circuito digital lido do arquivo BENCH, é criado um grafo de implicações específicas para cada porta lógica conforme as tabelas 3.1, 3.2, 3.3 e 3.4. Este grafo de implicações é utilizado para derivação das implicações realizadas como no *Vimplic*. Ao contrário do trabalho de Andrade et al. [2008b], a ferramenta implementada neste trabalho não salva em arquivo as novas implicações calculadas; elas são adicionadas no grafo de implicações que então é manipulado pelos algoritmos utilizados nas etapas de simplificação e hiper-resolução; tema abordado na próxima seção.

Tabela 3.1. Grafo de implicações gerado para as portas AND/NAND de duas entradas

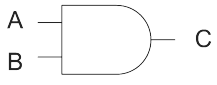
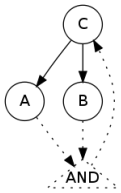
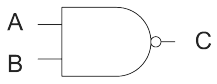
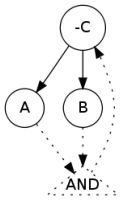
Porta lógica	Grafo de implicações criado
	
	

Tabela 3.2. Grafo de implicações gerado para as OR/NOR de duas entradas

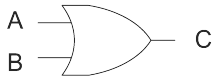
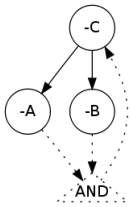
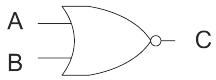
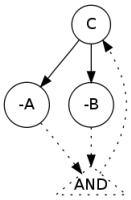
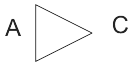
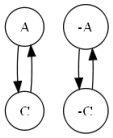
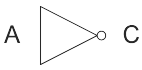
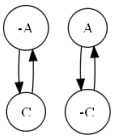
Porta lógica	Grafo de implicações criado
	
	

Tabela 3.3. Grafo de implicações gerado para todas as portas NOT/Buffer

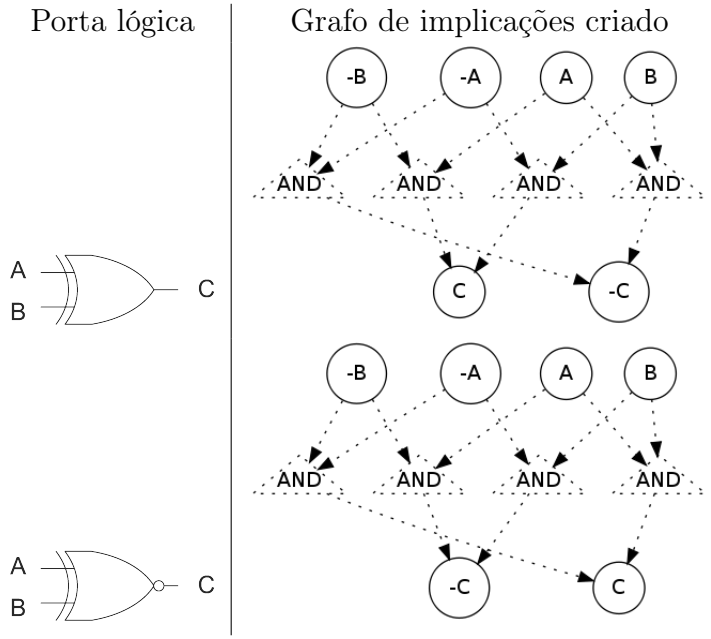
Porta lógica	Grafo de implicações criado
	
	

3.1.2 Etapa de redução de elementos equivalentes ativos

Uma vez construído o grafo de implicações e adicionadas as implicações redundantes propostas por Andrade et al. [2008b], se inicia a etapa de simplificações do grafo de implicações. Esta fase se inicia com a propagação de todos literais unitários, até que tal operação não possa mais ser realizada. Como alguns literais podem ser desativados, o grafo de implicações sofre redução se forem considerados apenas os elementos ativos.

Conforme visto na definição 2.3.2, se dois literais a e b implicam um ao outro, então um deles pode ser substituído pelo outro e em consequência simplificar ainda mais o grafo de implicações. O pré-processador busca por componentes fortemente conectados no grafo de implicações (procurando literais que implicam um ao outro) e marca todos os literais equivalentes como inativos, deixando ativo apenas o literal de menor módulo. Para manter a consistência da estrutura de dados do grafo de implicações - onde um nodo é considerado inativo apenas se ele próprio e todos seus antecedentes estão inativos - os nodos agora inativos são reorganizados de forma que têm em sua lista de descendentes o literal equivalente ativo, mas não têm em sua

Tabela 3.4. Grafo de implicações gerado para todas as portas XOR/XNOR.



lista de ascendentes nenhum literal ativo. Como a manipulação realizada no grafo de implicações desativando vértices equivalentes pode gerar novos literais unitários, então nova propagação unitária é realizada para potencialmente desativar novos literais e gerar nova simplificação.

Por fim, o algoritmo de hiper-resolução é aplicado e novas simplificações são realizadas. Como novos literais unitários podem ser produzidos e novos componentes fortemente conectados podem aparecer no grafo de implicações, é necessário executar tais passos novamente, junto com outra etapa de hiper-resolução. O processo é interrompido apenas quando o grafo não é modificado e o pré-processador termina sua execução.

3.1.3 Validação da ferramenta

Uma vez que a ferramenta deste trabalho foi implementada tendo como base o código-fonte das ferramentas *Vimplic* e *Hypre*, os módulos relativos a essas ferramentas foram considerados corretos e não foram revalidados.

O módulo de criação dos grafos de implicação, utilizado nas duas ferramentas, foi exaustivamente verificado a partir da especificação do circuito em formato BENCH. Para cada porta lógica listada no circuito de entrada o grafo de implicações correspondente foi verificado manualmente pelo autor, bem como possíveis interferências com implicações já geradas.

O processo de verificação se deu validando os grafos de implicação criados para circuitos simples, com poucas portas lógicas e analisando o resultado gerado pela ferramenta. À medida em que a validação se mostrou correta, circuitos mais complexos foram sendo utilizados para verificação e a associação das implicações referentes às várias portas lógicas permaneceu correta, o que, intuitivamente, leva a crer que o grafo de implicações gerado é correto para quaisquer circuitos cujo grafo possa ser gerado.

3.2 Exemplos de funcionamento da ferramenta

Esta seção mostra alguns testes didáticos realizados, mostrando passo a passo a evolução do grafo de implicações sob diferentes situações. A primeira situação, descrita em 3.2.1, mostra um caso onde a ferramenta consegue encontrar uma situação de satisfabilidade sem mesmo executar o resolvedor SAT. A segunda situação, descrita em 3.2.2, mostra um caso onde a ferramenta encontra um conflito nas implicações, significando insatisfabilidade. A seção 3.2.3 exibe uma situação onde o pré-processador não é capaz de definir ou não a satisfabilidade do problema, demandando a execução por um resolvedor SAT.

3.2.1 Funcionamento da ferramenta em uma situação de satisfabilidade

Seja o circuito mostrado na figura 3.2, criado a partir do teorema de DeMorgan e modificado propositalmente para que haja resultado satisfazível com o *miter* criado. Sejam também os grafos de implicação[1] mostrados nas figuras 3.3 a 3.9.

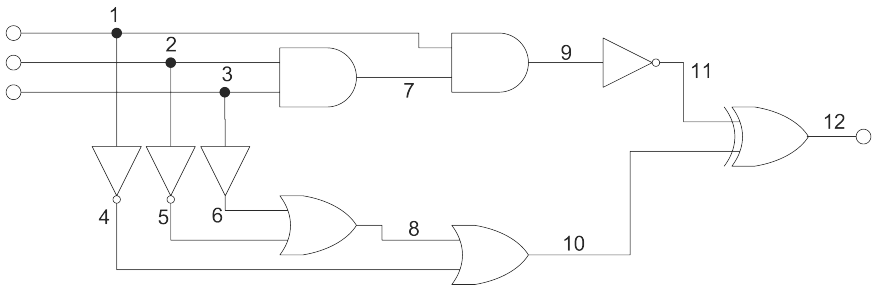


Figura 3.2. Circuito para exemplificar o funcionamento da ferramenta onde o resultado SAT é obtido.

[1]Nos grafos, os vértices representados por círculos cheios são variáveis ativas, vértices representados por retângulos vazios são variáveis desativadas. Se um vértice tem cor vermelha, significa literal com valor atribuído 0; se um vértice tem cor verde, o valor atribuído é 1; caso a cor do vértice seja cinza, o valor ainda é indefinido. Um literal barrado ($\overline{x}$) é representado nos grafos como $-x$. Arestas de conjunção "AND" são representadas por triângulos.

Inicialmente a ferramenta carrega o arquivo mostrado em A.1 (que descreve o circuito da figura 3.2), deriva as implicações de cada porta lógica conforme as tabelas 3.1 a 3.4 e adiciona as implicações propostas por Andrade et al. [2008b]. O resultado é o grafo de implicações mostrado na figura 3.3.

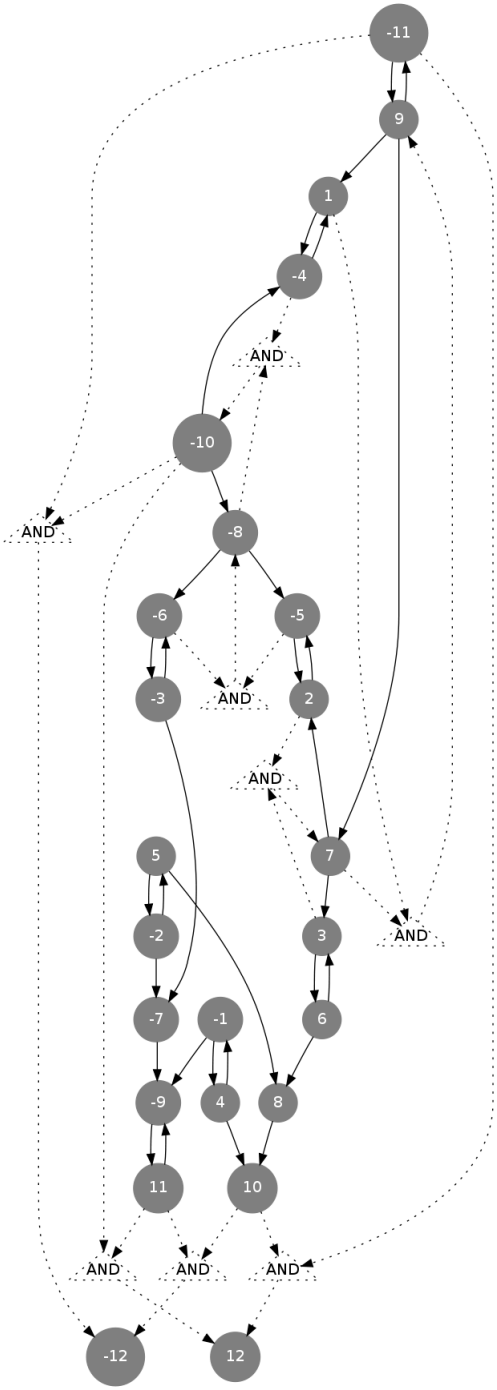


Figura 3.3. Grafo de implicações inicial para o circuito da figura 3.2

Uma vez que a satisfabilidade do *miter* deve ser verificada (deve-se encontrar

uma entrada do circuito que leve sua saída para verdadeiro), é adicionado ao processo de verificação uma cláusula unitária que diz que o sinal 12 deve possuir valor booleano verdadeiro. A adição desta propriedade faz com que os sinais 12 e $\overline{12}$ sejam definidos e portanto, desativados. Pelo fato de 12 e $\overline{12}$ estarem desativados, as implicações geradas pela por XOR do *miter* são modificadas e então $\overline{10} \rightarrow 11$ e $10 \rightarrow \overline{11}$ (e contra-positivas) são adicionadas, conforme visto no grafo da figura 3.4.

Efetuadas as propagações unitárias (no exemplo, apenas do sinal 12), a ferramenta detecta todos os componentes fortemente conectados no grafo de implicações, deixando ativos apenas os sinais de menor módulo que fazem parte do componente. A figura 3.5 mostra a grande quantidade de sinais equivalentes encontrados e desativados (marcados com um retângulo vazio), principalmente se comparados com o grafo mostrado na figura 3.4.

Para manter a propriedade de que um nodo é considerado inativo apenas se ele próprio e todos seus antecedentes estão inativos, os sinais desativados são remanejados, apontando para seus equivalentes. Esta situação é mostrada na figura 3.6.

Os vértices "AND" estão ativos apenas se todos os vértices que nele chegam saem de vértices de sinal ativos. Esta etapa efetua a eliminação de arestas de conjunção (vértices "AND") desativados através de análise das implicações e propagações. A partir do ponto em que tais vértices "AND" são eliminados não é mais possível estabelecer relações diretas entre o grafo de implicações e o circuito digital original. A figura 3.7 mostra o grafo de implicações após a eliminação de tais vértices.

Após todas as simplificações realizadas, é feita a hiper-resolução. As implicações $8 \rightarrow 9$ e $8 \rightarrow \overline{9}$ eliminam o literal 8 e seu contra-positivo $\overline{8}$, gerando o grafo da figura 3.8 (dada a simplicidade do exemplo didático mostrado no texto, esta etapa de hiper-resolução é equivalente à resolução binária; em circuitos mais complexos simplificações mais poderosas podem ocorrer).

Estando modificado o grafo de implicações, novas propagações unitárias são realizadas, nova detecção de componentes fortemente conectados e novas hiper-resoluções devem ser aplicadas. No exemplo mostrado não é possível efetuar propagações unitárias e não aparecem novos componentes fortemente conectados. Uma hiper-resolução, no entanto, pode ser aplicada. As implicações $\overline{2} \rightarrow 9$ e $\overline{2} \rightarrow \overline{9}$; bem como $\overline{1} \rightarrow 9$ e $\overline{1} \rightarrow \overline{9}$ são eliminadas em um único passo, deixando apenas os literais 9 e $\overline{9}$ ativos, conforme visto no grafo da figura 3.9.

A partir deste ponto nenhuma regra de simplificação pode ser aplicada e o pré-processador interrompe sua execução, gerando um arquivo CNF DIMACS mostrando as atribuições das entradas que levam à situação.

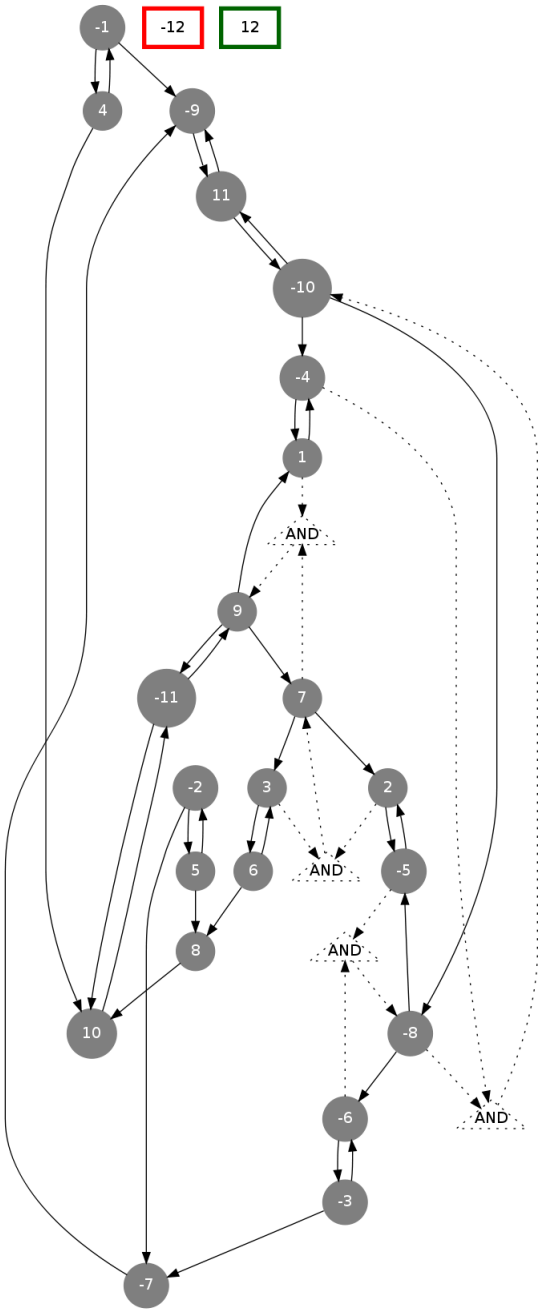


Figura 3.4. Grafo de implicações após a propagação dos literais 12 e $\overline{12}$ (devido à propriedade). As implicações $\overline{10} \rightarrow 11$ e $10 \rightarrow \overline{11}$ (e suas contra-positivas) foram adicionadas.

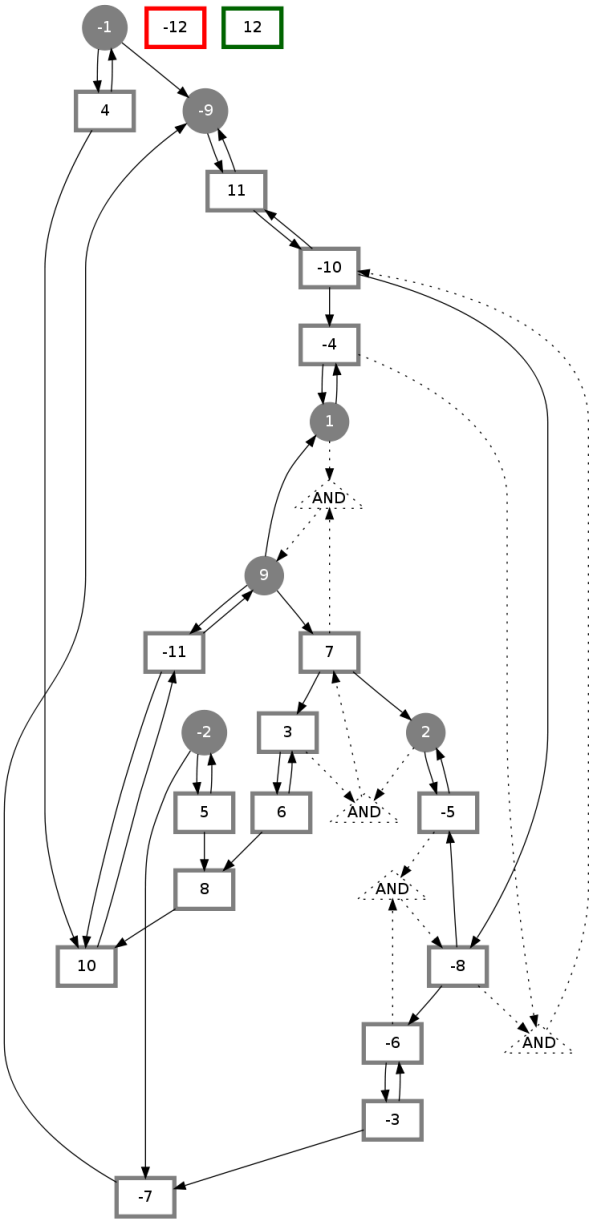


Figura 3.5. Grafo de implicações após a detecção de componentes fortemente conectados.

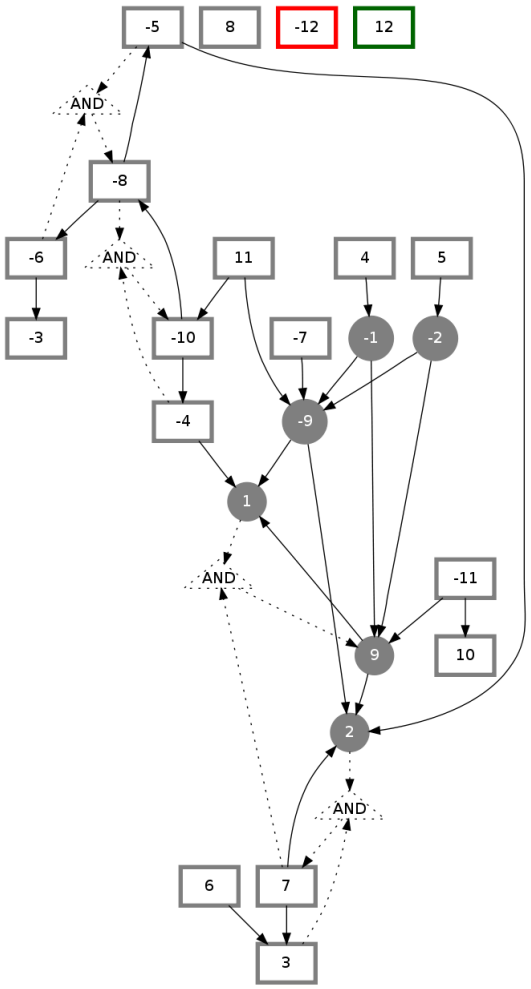


Figura 3.6. Grafo de implicações após a desativação de vértices que faziam parte de componentes fortemente conectados.

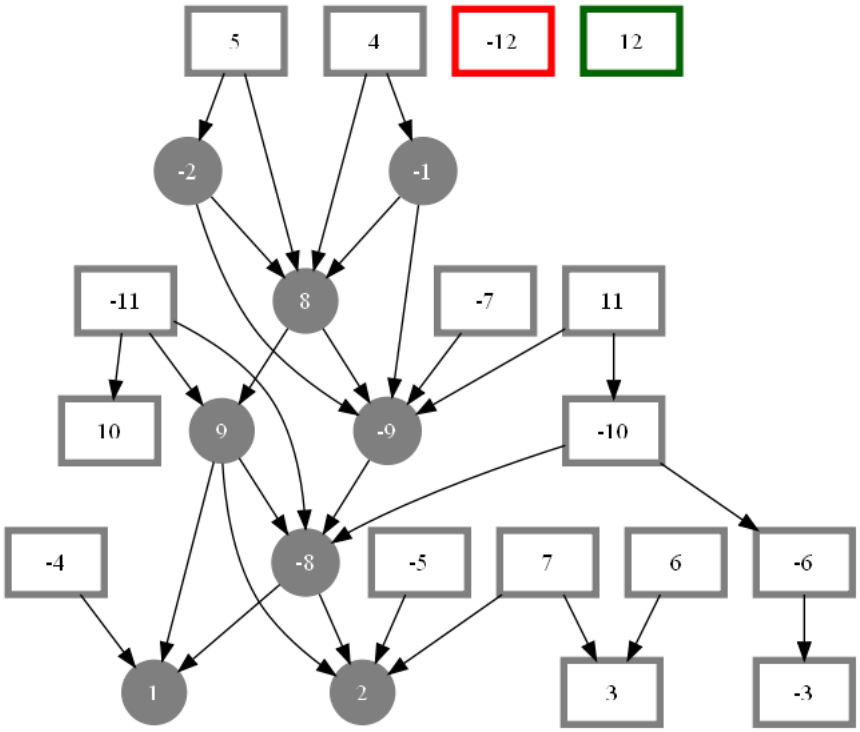


Figura 3.7. Grafo de implicações após a remoção de vértices AND.

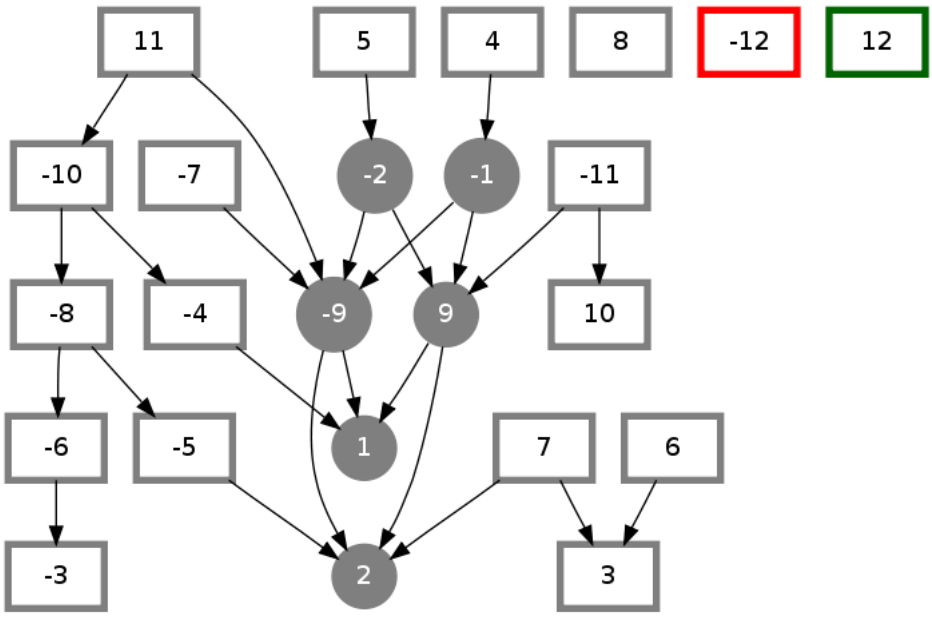


Figura 3.8. Grafo de implicações após a primeira hiper-resolução.

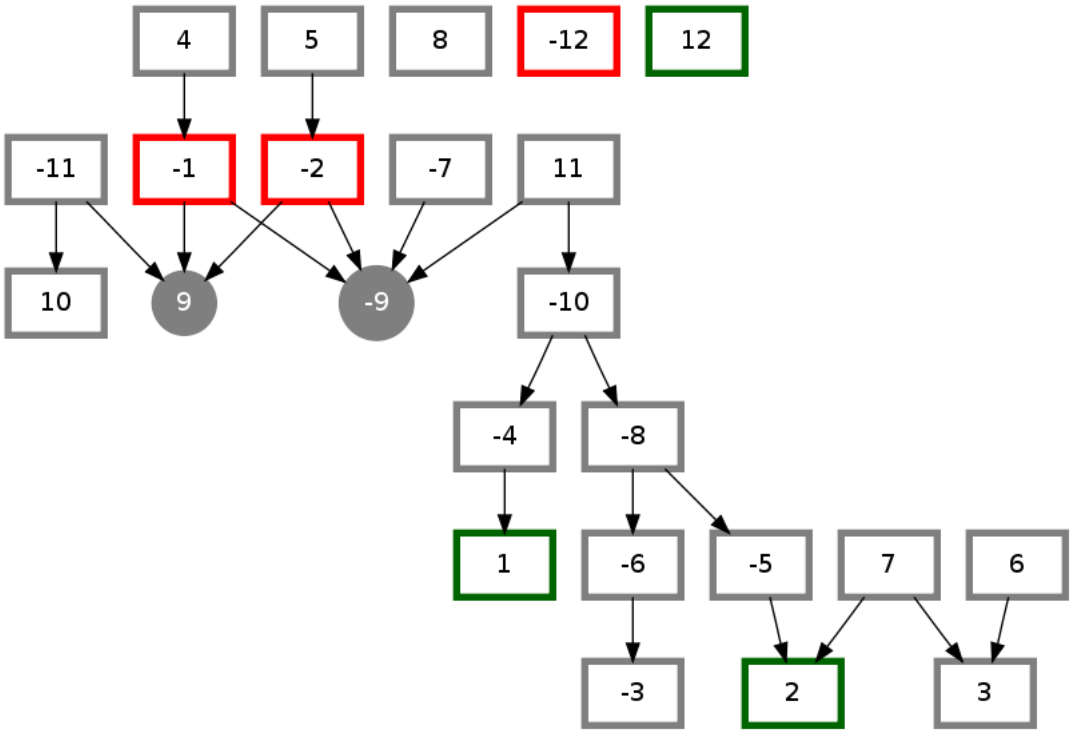


Figura 3.9. Grafo de implicações após a segunda hiper-resolução. A partir deste ponto o grafo fica estável.

3.2.2 Funcionamento da ferramenta em uma situação de insatisfabilidade

Seja o circuito mostrado na figura 3.10, criado a partir do teorema de DeMorgan e modificado propositalmente para que haja resultado insatisfazível com o *miter* criado. Sejam também os grafos de implicação mostrados nas figuras 3.11 a 3.15.

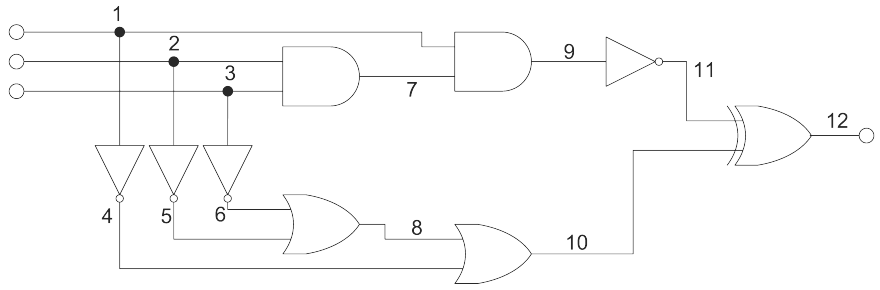


Figura 3.10. Circuito para exemplificar o funcionamento da ferramenta que produz uma situação UNSAT.

Inicialmente a ferramenta carrega o arquivo mostrado em A.2 (que descreve o circuito da figura 3.10), deriva as implicações de cada porta lógica conforme as tabelas 3.1 a 3.4 e adiciona as implicações propostas por Andrade et al. [2008b]. O resultado é o grafo de implicações mostrado na figura 3.11.

Uma vez que a satisfabilidade do *miter* deve ser verificada (deve-se encontrar uma entrada do circuito que leve sua saída para verdadeiro), é adicionado ao processo de verificação uma cláusula unitária que diz que o sinal 12 deve possuir valor booleano verdadeiro. A adição desta propriedade faz com que os sinais 12 e $\overline{12}$ sejam definidos e portanto, desativados. Pelo fato de 12 e $\overline{12}$ estarem desativados, as implicações geradas pela por XOR do *miter* são modificadas e então $\overline{10} \rightarrow 11$ e $10 \rightarrow \overline{11}$ (e contra-positivas) são adicionadas, conforme visto no grafo da figura 3.12.

Efetuadas as propagações unitárias (no exemplo, apenas do sinal 12), a ferramenta detecta todos os componentes fortemente conectados no grafo de implicações, deixando ativos apenas os sinais de menor módulo que fazem parte do componente. A figura 3.13 mostra a grande quantidade de sinais equivalentes encontrados e desativados (marcados com um retângulo vazio), principalmente se comparados com o grafo mostrado na figura 3.12.

Para manter a propriedade de que um nodo é considerado inativo apenas se ele próprio e todos seus antecedentes estão inativos, os sinais desativados são remanejados, apontando para seus equivalentes. Esta situação é mostrada na figura 3.14.

Os vértices "AND" estão ativos apenas se todos os vértices que nele chegam saem de vértices de sinal ativos. Esta etapa efetua a eliminação de arestas de conjunção

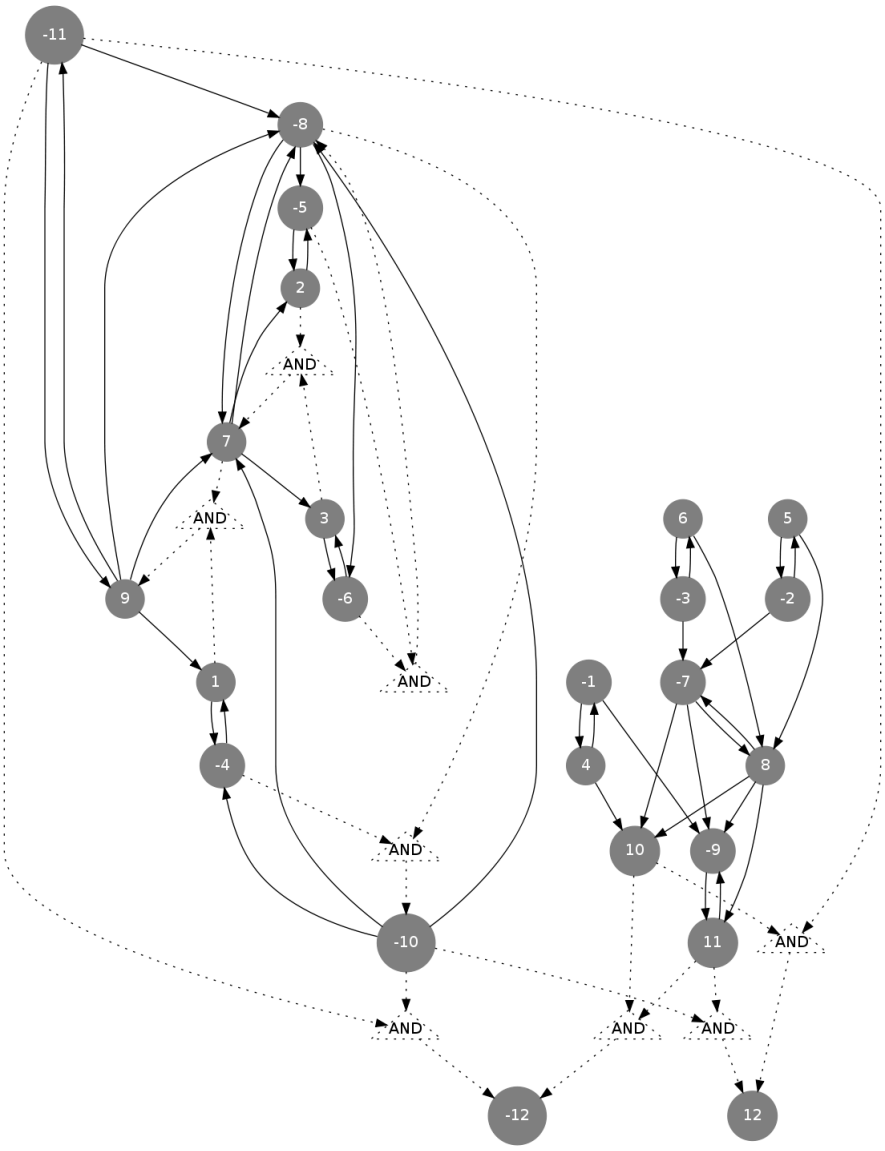


Figura 3.11. Grafo de implicações inicial para o circuito da figura 3.10

(vértices "AND") desativados através de propagações. Durante a propagação do sinal $\overline{7}$ a ferramenta detecta um sinal 7 sendo sucessor do primeiro. Em outras palavras, é encontrada uma implicação $\overline{7} \rightarrow 7$, o que é uma contradição. Neste ponto a ferramenta termina, mostrando uma mensagem informado que há uma contradição entre os dois sinais, tornando impossível a satisfabilidade do circuito. A figura 3.15 mostra o grafo de implicações onde é possível notar a contradição provocada por $\overline{7} \rightarrow 9 \rightarrow 7$. O arquivo DIMACS CNF não é gerado.

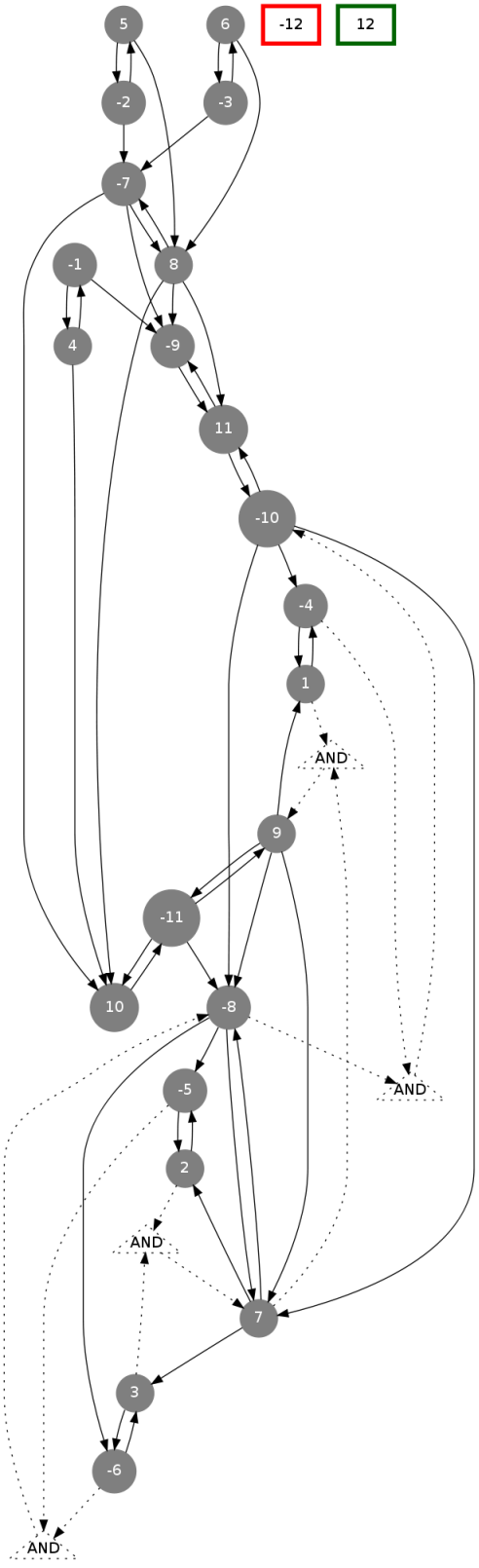


Figura 3.12. Grafo de implicações após a propagação dos literais 12 e $\overline{12}$ (devido à propriedade). As implicações $\overline{10} \rightarrow 11$ e $10 \rightarrow \overline{11}$ (e suas contra-positivas) foram adicionadas.

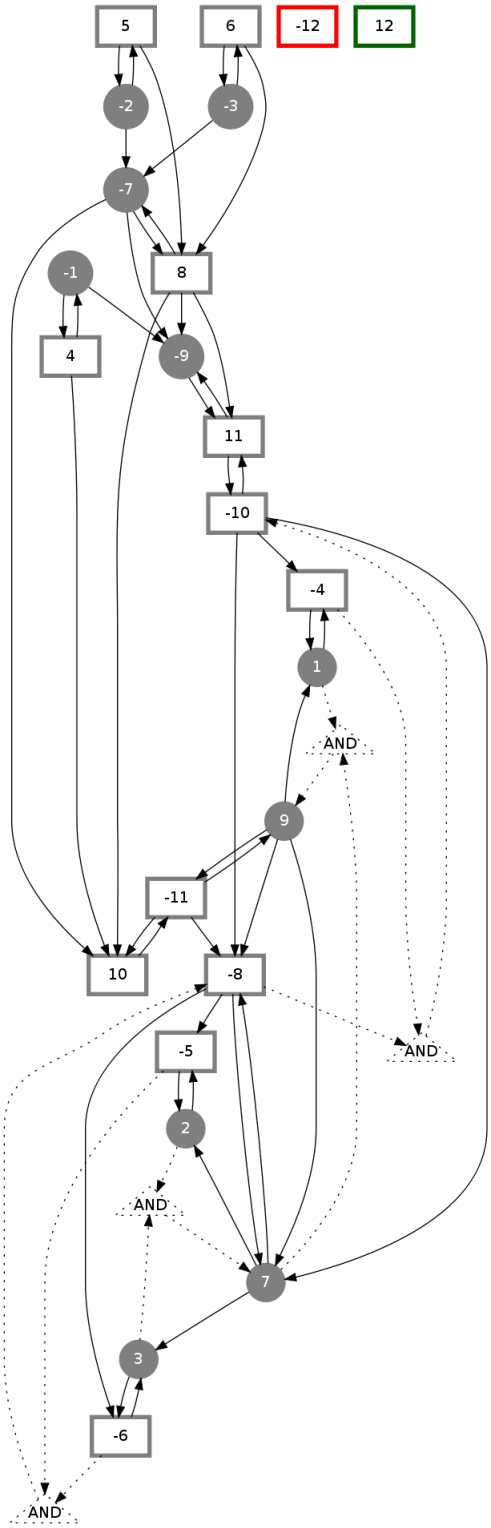


Figura 3.13. Grafo de implicações após a detecção de componentes fortemente conectados.

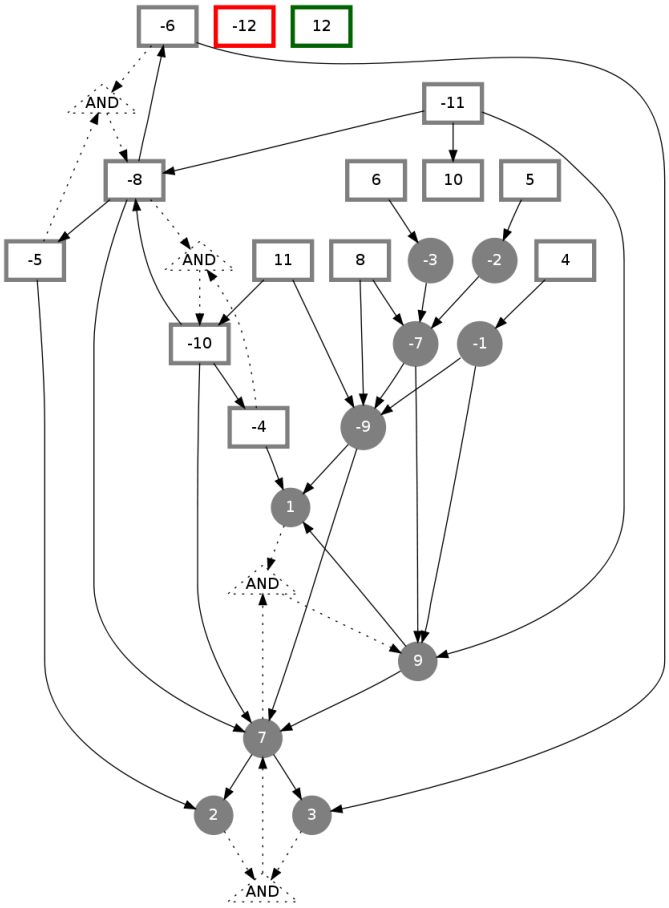


Figura 3.14. Grafo de implicações após a desativação de vértices que faziam parte de componentes fortemente conectados.

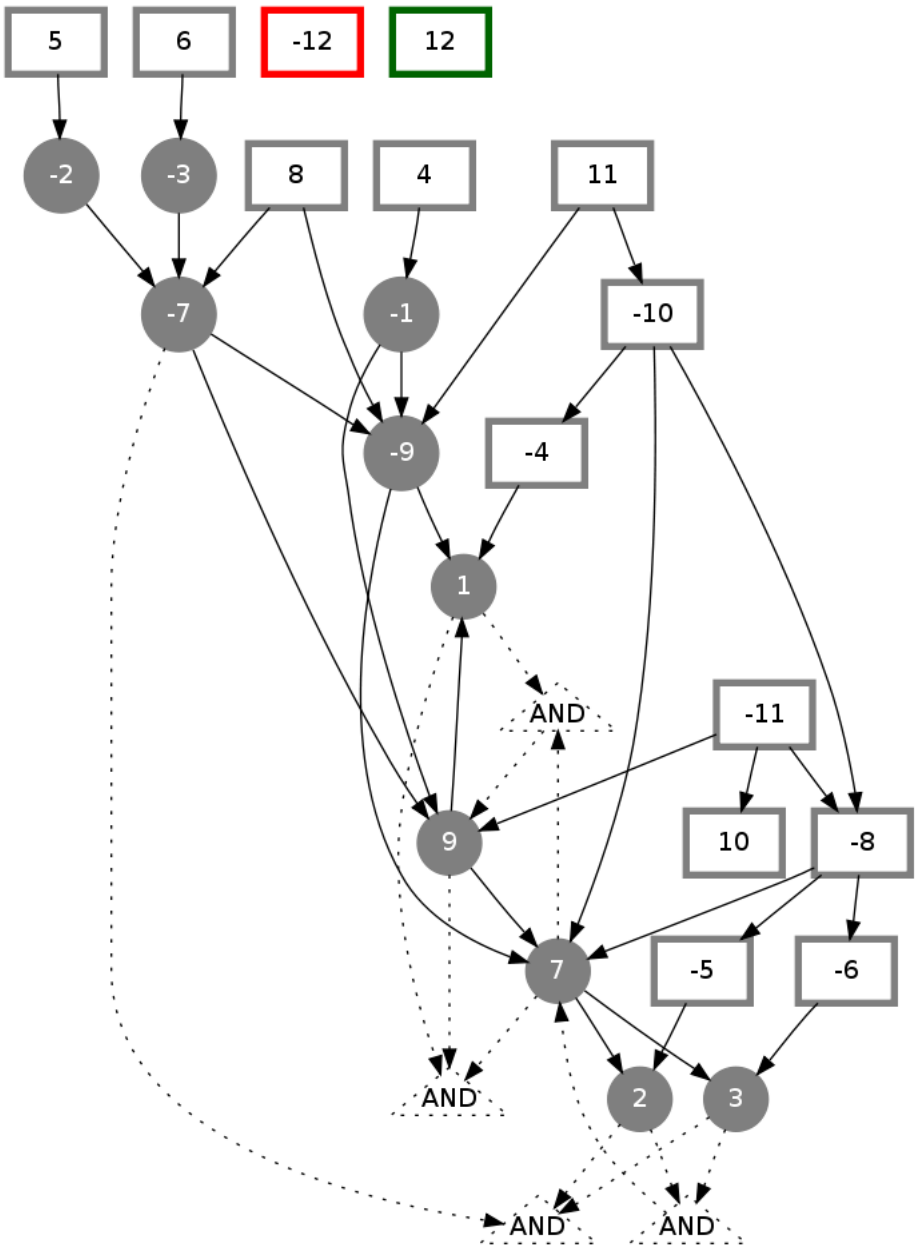


Figura 3.15. Grafo de implicações de simplificação de vértices AND. Há um problema, pois $\overline{7} \to 9 \to 7$, o que é uma contradição.

3.2.3 A ferramenta como meio de simplificação do problema

Caso a ferramenta implementada neste trabalho não encontre conflitos ou chegue a detectar a satisfabilidade do circuito a ser verificado durante as etapas de simplificação (interrompidas pela impossibilidade de continuar aplicando as regras já vistas anteriormente), é realizada uma varredura entre os vértices ainda ativos presentes no grafo de implicações, gerando cláusulas CNF em um arquivo. Cada implicação $A \rightarrow B$ encontrada no grafo é mapeada para uma cláusula CNF do tipo $(\neg A \vee B)$ na saída, que segue o padrão CNF DIMACS, e que pode aplicada a resolvedores SAT correntes como o CryptoMiniSat [Soos et al., 2009], HaifaSat [Gershman, 2005] ou BerkMin [Goldberg & Novikov, 2002].

É importante ressaltar que, mesmo que conflitos ou a satisfabilidade não tenham sido encontrados pelo pré-processador, a situação comum é que as regras de simplificação reduzam em grande parte o problema a ser tratado. A próxima seção mostra alguns testes realizados nesse sentido, comprovando que, mesmo em casos onde o resolvedor SAT não possa ser dispensado do processo de verificação, o tempo total envolvendo a execução do pré-processador e a execução do resolvedor SAT ainda é menor que no caso onde o resolvedor SAT é aplicado de forma direta.

Capítulo 4

Testes e resultados

Este capítulo é dedicado a mostrar os testes realizados com a ferramenta Hiperplic e à análise dos resultados obtidos.

4.1 Testes

Para efetuar a verificação de equivalência de dois circuitos quaisquer utilizando esta ferramenta são necessários os seguintes passos:

1. Descrição do circuito de referência no formato BENCH;
2. Descrição do circuito sob verificação no formato BENCH;
3. Criação de um *miter* utilizando o circuito de referência e o circuito sob verificação, também no formato BENCH;
4. Submissão do *miter* criado à ferramenta, que irá executar o pré-processamento gerando implicações adicionais;
5. Submissão do *miter* criado à ferramenta, que irá executar o pré-processamento sem gerar implicações adicionais;
6. Execução de resolvedores SAT sobre a saídas geradas pela ferramenta em cada uma das duas situações anteriores;
7. Coleta das informações sobre tempo de execução do processo de verificação.

Todos os testes foram executados sobre *miters* descritos no formato BENCH já mencionado e a propriedade de saída do *miter* é adicionada automaticamente pelo pré-processador. Todos os *miters* foram criados utilizando dois circuitos idênticos.

4.1.1 Utilização da ferramenta implementada

Para a execução da ferramenta implementada neste trabalho, é necessário se utilizar de uma interface de linha de comando, como um *shell* do sistema Linux, por exemplo. O comando para execução da ferramenta é o seguinte:

```
hiperplic [-i] -e arquivo-entrada -s arquivo-saida > registro-temporizacao
```

As opções `-e`, `-s` e `-i` têm a seguinte função:

- `-e`: define um nome de arquivo de entrada no formato BENCH; é uma opção obrigatória.
- `-s`: define um nome de arquivo de saída, no formato CNF DIMACS; é uma opção obrigatória.
- `-i`: inibe a geração de implicações adicionais no circuito, executando apenas a hiper-resolução.

Durante a execução dos testes, a saída de temporização informada pela ferramenta `hiperplic` foi redirecionada para um arquivo para que os resultados pudessem ser analisados posteriormente por uma ferramenta de coleta de dados escrita especificamente para este fim. Esta ferramenta coleta os resultados de cada resolvedor SAT e da ferramenta Hiperplic para cada circuito e monta uma tabela no formato CSV, que pode ser lido por planilhas eletrônicas como o Gnumeric, OpenOffice Calc ou Microsoft Excel.

4.1.2 Resolvedores SAT utilizados nos testes

Para comparação de desempenho do processo de verificação combinatória, foram feitos testes com os seguintes resolvedores SAT (todos eles com o tempo de execução limitado a 10800 segundos - tempo equivalente a 3 horas):

- BerkMin 5.6.1[Goldberg & Novikov, 2002]: um resolvedor SAT relativamente antigo e de código proprietário, mas que obtém resultados satisfatórios na resolução de problemas SAT até então. Foi executado com a opção `s 1`, que utiliza heurísticas voltadas para a área de verificação de equivalência.
- CryptoMiniSat 2.5.1 [Soos et al., 2009]: um resolvedor SAT bastante recente, vencedor do torneio SAT Race, 2010. É capaz de determinar automaticamente instâncias de SAT industriais (o caso deste trabalho) ou criptográficas e aplicar heurísticas adequadas a cada caso. Foi executado sem parâmetros especiais.

- HaifaSat 1.0 [Gershman, 2005]: um resolvedor SAT que utiliza pré-processamento com hiper-resolução. Foi executado sob duas variantes, uma que executava o pré-processamento com hiper-resolução e outra sem a aplicação de hiper-resolução (parâmetros `-p 1` e `-p 0`, respectivamente).

4.1.3 Computadores utilizados nos testes

Os testes foram executados paralelamente em três computadores diferentes, com o objetivo de agilizar a obtenção de resultados. Para que não houvessem erros de comparação de tempo execução, testes com o mesmo tipo de circuito foram feitos no mesmo computador. A configuração de cada um dos computadores usados é a seguinte:

- Computador 1:
  - Processador: AMD Athlon IV 3.0GHz com núcleo duplo;
  - Memória RAM: 2 GB DDR2;
  - Sistema operacional: Ubuntu Linux 10.04.
- Computador 2:
  - Processador: Intel Core 2 Quad Q6600 2.4 GHz (núcleo quádruplo);
  - Memória RAM: 8 GB DDR2;
  - Sistema operacional: Linux Fedora Core 7.
- Computador 3:
  - Processador: AMD Sempron 1.8GHz com núcleo simples;
  - Memória RAM: 512 MB DDR;
  - Sistema operacional: Ubuntu Linux 10.04.

4.1.4 Circuitos utilizados nos testes

Os circuitos listados abaixo foram utilizados nos testes com a ferramenta `hiperplic`. Com exceção dos multiplicadores e dos testes ISCAS'85, os circuitos foram gerados pela ferramenta de Andrade et al. [2008a]. Em todos os casos foram criados *miters* de circuitos idênticos (portanto o resultado para o problema de satisfabilidade booleana é UNSAT).

1. Circuitos dos testes ISCAS'85;

2. Multiplicadores Wallace idênticos, com tamanhos variados;
3. Multiplicadores Dadda idênticos, com tamanhos variados;
4. Multiplicadores Reduced idênticos, com tamanhos variados;
5. Multiplicadores baseados em somadores *carry save* idênticos, com tamanhos variados;
6. *Barrel shifters* idênticos, com tamanhos variados;
7. Somadores idênticos do tipo *block carry lookahead*, com tamanhos variados;
8. Somadores idênticos do tipo *ripple carry adder*, com tamanhos variados;
9. Somadores idênticos do tipo *carry skip adder*, com tamanhos variados;
10. Somadores idênticos do tipo *carry save adder*, com tamanhos variados;
11. Somadores idênticos do tipo *carry select adder*, com tamanhos variados;

Combinando cada um dos 11 circuitos mostrados acima com as duas opções de execução do Hiperplic (mostradas na seção 4.1.1), os testes de referência (execução dos resolvedores SAT sem aplicação de Hiperplic) e os quatro resolvedores SAT (considerando que o resolvedor HaifaSat foi executado duas vezes, conforme descrito na seção 4.1.2), foram executados 132 conjuntos de testes. Cada um desses conjuntos envolve a aplicação de um tipo de pré-processamento e um resolvedor a várias instâncias de um tipo de circuito miter (as diferentes instâncias variam o tamanho daquele tipo de *miter*). A seção 4.2 mostra tabelas que sumarizam os resultados destes testes e faz uma análise dos dados obtidos.

4.2 Resultados

Esta seção mostra os resultados obtidos nos testes realizados. As tabelas 4.1 e 4.2 mostram os resultados obtidos na aplicação da ferramenta Hiperplic nos circuitos de referência do conjunto de *benchmarks* propostos por Brglez & Fujiwara [1985]. A coluna "# circuito" indica o número do identificador do circuito ISCAS'85 e as demais colunas serão descritas a seguir.

Nas tabelas de número 4.3 a 4.22, a coluna nomeada "T.c." indica o tamanho do circuito *miter* criado. Por exemplo, o número 16 na coluna "T.c." da tabela 4.3 indica que o teste realizado com um *miter* formado por dois multiplicadores Wallace

idênticos, cada um deles capaz de multiplicar duas variáveis de 16 bits cada e gerar um resultado de 16 bits. As demais colunas, presentes em todas as tabelas, são descritas a seguir:

- **BM**: colunas referentes ao resolvedor SAT Berkmin;
- **CMS**: colunas referentes ao resolvedor SAT CryptoMiniSat;
- **HF**: colunas referentes ao resolvedor SAT HaifaSat executado sem pré-processamento;
- **HFP**: colunas referentes ao resolvedor SAT HaifaSat executado com pré-processamento interno baseado em hiper-resolução.
- **Tempo referência**: tempo de referência do teste, obtido pela aplicação do resolvedor SAT sem utilização da ferramenta Hiperplic;
- **T. pré**: tempo de execução, em segundos, do pré-processamento realizado pela ferramenta Hiperplic;
- **Tempo total verif.**: tempo total da verificação do circuito, em segundos. Estas colunas mostram o tempo de execução dos resolvedores SAT supracitados sobre o arquivo pré-processado pelo Hiperplic, somado com o tempo de pré-processamento mostrado na coluna "T. pré";
- **Ganho**: ganho de tempo total de verificação graças à aplicação da ferramenta Hiperplic. Mostra a redução percentual do tempo de verificação total comparado com o tempo de referência, obtido pela execução do resolvedor SAT sem a aplicação de Hiperplic.

Os resultados marcados como "-" nas tabelas indicam impossibilidade de cálculo de ganho de performance ou erro numérico na coleta de tempos de execução. Campos marcados com "> 3h" indicam que a verificação foi interrompida por levar mais tempo que os 10800 segundos definidos como limite; campos marcados como "Indef." indicam que houve um ganho de performance na aplicação da ferramenta implementada neste trabalho, mas não foi possível calcular o ganho percentual (normalmente porque a verificação foi interrompida após as 3 horas de execução).

Tabela 4.1. Desempenho obtido na verificação de circuitos integrantes do ISCAS'85. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

#	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
circuito	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
432	0	0,02	0,01	0,01	0,09	0,09	0,09	0,09	0,09	-	-340,03%	-780,05%	-780,05%
499	0,05	0,08	0,05	0,06	0,10	0,13	0,13	0,16	0,17	-160,01%	-62,51%	-220,01%	-183,34%
880	0,04	0,39	0,27	0,11	0,14	0,14	0,14	0,14	0,14	-250,02%	64,10%	48,14%	-27,28%
1355	0,13	0,15	0,2	0,08	0,25	0,25	0,25	0,25	0,25	-93,86%	-68,01%	-26,01%	-215,02%
1908	0,16	0,26	0,43	0,46	0,41	0,41	0,41	0,41	0,41	-157,52%	-58,47%	4,18%	10,43%
2670	0,35	0,2	0,8	0,33	0,47	0,47	0,47	0,47	0,47	-34,87%	-136,02%	41,00%	-43,04%
3540	2,53	3,11	20,35	18,62	1,73	1,73	1,73	1,73	1,73	31,54%	44,31%	91,49%	90,70%
5315	3,66	1	7,07	4,63	1,12	1,12	1,12	1,12	1,12	69,29%	-12,41%	84,10%	75,72%
6288	4289,21	248,97	$> 3h$	$> 3h$	1,22	1,22	1,22	1,22	1,22	99,97%	99,51%	Indef.	Indef.
7552	10,03	1,68	24,25	20,38	2,04	2,04	2,04	2,04	2,04	79,62%	-21,67%	91,57%	89,97%

Tabela 4.2. Desempenho obtido na verificação de circuitos integrantes do ISCAS'85. O pré-processamento aplicado envolveu apenas hiper-resolução.

#	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
circuito	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
432	0	0,02	0,01	0,01	0,07	0,07	0,07	0,07	0,07	-	-240,02%	-580,03%	-580,03%
499	0,05	0,08	0,05	0,06	0,08	0,11	0,11	0,14	0,15	-112,01%	-32,51%	-172,01%	-143,34%
880	0,04	0,39	0,27	0,11	0,10	0,10	0,10	0,10	0,10	-150,02%	74,36%	62,96%	9,09%
1355	0,13	0,15	0,2	0,08	0,16	0,16	0,16	0,16	0,16	-26,16%	-9,34%	17,99%	-105,01%
1908	0,16	0,26	0,43	0,46	0,20	0,20	0,20	0,20	0,20	-27,51%	21,53%	52,56%	55,65%
2670	0,35	0,2	0,8	0,33	0,25	0,25	0,25	0,25	0,25	28,00%	-26,01%	68,50%	23,63%
3540	2,53	3,11	20,35	18,62	0,67	0,67	0,67	0,67	0,67	73,60%	78,52%	96,72%	96,41%
5315	3,66	1	7,07	4,63	0,68	0,68	0,68	0,68	0,68	81,31%	31,60%	90,32%	85,23%
6288	4289,21	248,97	$> 3h$	$> 3h$	1,08	1,08	1,08	1,08	1,08	99,97%	99,57%	Indef.	Indef.
7552	10,03	1,68	24,25	20,38	1,23	1,23	1,23	1,23	1,23	87,72%	26,66%	94,92%	93,95%

4.2.1 Circuitos ISCAS'85

As tabelas 4.1 e 4.2 mostram os resultados obtidos nos testes com instâncias dos circuitos ISCAS'85, executados no computador 3, citado na seção 4.1.3.

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias ISCAS'85 foi obtido pelo resolvedor Berkmin, apesar de ser um resolvedor relativamente antigo. Outro resultado interessante é que, com exceção do circuito de número 499, os problemas de verificação foram resolvidos diretamente pelo pré-processador Hiperplic, que encontrou conflitos nessas situações.

Comparando o tempo total de verificação com os resultados obtidos pelo resolvedor de melhor desempenho, pode-se observar que o pré-processador é mais lento nas instâncias menores, mas consegue resolver o problema mais rapidamente nas cinco maiores instâncias do ISCAS'85 quando as implicações adicionais não foram utilizadas (ver tabela 4.2).

Outra comparação relevante é entre os tempos totais de verificação obtidos quando a geração de implicações extras foi usada (tabela 4.1) e quando não foi usada (tabela 4.2): a utilização das implicações adicionais aumenta o tempo total de verificação e gera efetivamente os mesmos efeitos obtidos quando apenas a hiper-resolução

foi utilizada, mostrando que, nas instâncias ISCAS'85, a hiper-resolução é suficiente no pré-processamento.

4.2.2 Circuitos multiplicadores

Esta seção mostra o resultado dos testes obtidos na verificação de multiplicadores. As instâncias desses circuitos foram limitadas a multiplicadores com operandos de até 32 bits, que, em alguns casos, não puderam ser verificados no tempo limite de 3 horas de execução. Na maioria dos casos as instâncias de multiplicadores usadas nos testes envolvem circuitos de tamanho comercial, como 8, 16 e 32 bits.

4.2.2.1 Multiplicadores Wallace

As tabelas 4.3 e 4.4 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois multiplicadores Wallace. Estes testes foram executados no computador 2, citado na seção 4.1.3.

Tabela 4.3. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Wallace de tamanhos idênticos. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0,00	0,00	0,00	0,00	-	-	-	-	-	-	-	-	-
2	0,00	0,00	0,00	0,00	-	-	-	-	-	-	-	-	-
4	0,00	0,01	0,01	0,01	0,05	0,05	0,06	0,05	0,05	-	-529,91%	-429,91%	-429,91%
8	22,15	14,72	69,77	63,17	0,76	5,89	11,32	45,06	52,45	73,40%	23,09%	35,42%	16,97%
12	191,87	736,57	> 3h	5550,74	6,51	19,49	563,27	3216,65	2921,59	89,84%	23,53%	Indef.	47,37%
16	738,26	-	> 3h	> 3h	34,03	82,03	129,45	10363,39	10030,28	88,89%	-	Indef.	Indef.
20	2670,11	6591,35	> 3h	> 3h	123,79	327,83	263,42	> 3h	> 3h	87,72%	96,00%	-	-
24	4309,18	-	> 3h	> 3h	420,13	843,39	815,57	> 3h	> 3h	80,43%	-	-	-
28	8245,72	297,68	> 3h	> 3h	1080,08	1890,03	1553,90	> 3h	> 3h	77,08%	-422,00%	-	-
32	> 3h	621,96	> 3h	> 3h	2445,45	3752,68	3165,80	> 3h	> 3h	Indef.	-409,00%	-	-

Tabela 4.4. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Wallace de tamanhos idênticos. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0,00	0,00	0,00	0,00	-	-	-	-	-	-	-	-	-
2	0,00	0,00	0,00	0,00	-	-	-	-	-	-	-	-	-
4	0,00	0,01	0,01	0,01	0,04	0,04	0,05	0,04	0,04	-	-369,95%	-269,95%	-269,95%
8	22,15	14,72	69,77	63,17	0,09	2,17	11,49	57,86	46,74	90,19%	21,92%	17,07%	26,00%
12	191,87	736,57	> 3h	5550,74	0,21	14,42	86,41	3592,95	4749,74	92,49%	88,27%	Indef.	14,43%
16	738,26	-	> 3h	> 3h	0,42	53,97	75,68	> 3h	10402,48	92,69%	-	-	Indef.
20	2670,11	6591,35	> 3h	> 3h	0,80	189,16	94,07	> 3h	> 3h	92,92%	98,57%	-	-
24	4309,18	-	> 3h	> 3h	1,42	378,88	337,73	> 3h	> 3h	91,21%	-	-	-
28	8245,72	297,68	> 3h	> 3h	2,29	732,56	413,05	> 3h	> 3h	91,12%	-38,76%	-	-
32	> 3h	621,96	> 3h	> 3h	3,69	1543,86	625,45	> 3h	> 3h	Indef.	-0,56%	-	-

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias de multiplicadores Wallace foi obtido pelo resolvedor CryptoMiniSAT, um resolvedor SAT apresentado em agosto de 2010 e vencedor do campeonato de resolvedores SATRACE'2010.

Ao contrário do que aconteceu nos testes com os circuitos ISCAS'85, em nenhum caso o pré-processador Hiperplic resolveu sozinho o problema de verificação combinatória, mas conseguiu reduzir o tempo de resolução do problema em 17 das 22 instâncias mostradas.

As tabelas 4.3 e 4.4 mostram que, nas pequenas instâncias (de tamanho 1 e 2), o pré-processador e os resolvedores apresentaram tempo de execução muito baixos ou zero, dificultando a análise - estes dados não serão incluídos na análise. Já os resultados obtidos nas instâncias maiores mostram que a aplicação da ferramenta Hiperplic foi vantajosa na maioria dos casos: apenas no *miter* de dois multiplicadores 28x28 o resolvedor CryptoMiniSAT apresentou piora significativa no tempo verificação quando se compara a execução com e sem a utilização da ferramenta Hiperplic (se considerarmos a não utilização das implicações adicionais).

Outra comparação relevante é entre os tempos totais de verificação obtidos quando a geração de implicações extras foi usada (tabela 4.3) e quando não foi usada (tabela 4.4): a utilização das implicações adicionais aumenta o tempo total de verificação e gera um grande impacto negativo no tempo de verificação total, o que permite concluir, novamente, que a hiper-resolução é suficiente no pré-processamento de multiplicadores Wallace.

4.2.2.2 Multiplicadores Dadda

As tabelas 4.5 e 4.6 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois multiplicadores Dadda. Estes testes foram executados no computador 1, citado na seção 4.1.3.

Tabela 4.5. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Dadda de tamanhos idênticos. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0,01	0	0	0,03	-	-	-	-	-	-	-	-
2	0	0,01	0	0	0,03	-	-	-	-	-	-	-	-
4	0,01	0,01	0,01	0	0,06	0,07	0,07	0,06	0,07	-600,00%	-600,00%	-500,00%	-
8	98,13	12,29	249,78	224,33	0,81	27,71	5,74	195,65	189,01	71,76%	53,30%	21,67%	15,74%
16	5375,35	156,11	> 3h	> 3h	42,8	423,28	212,71	9760,49	7026,91	92,13%	-36,26%	Indef.	Indef.
32	> 3h	1779,07	> 3h	> 3h	3536,79	9817,39	4711,67	> 3h	> 3h	Indef.	-164,84%	-	-

Tabela 4.6. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Dadda de tamanhos idênticos. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0,01	0	0	0,04	-	-	-	-	-	-	-	-
2	0	0,01	0	0	0,03	-	-	-	-	-	-	-	-
4	0,01	0,01	0,01	0	0,05	0,06	0,06	0,05	0,06	-500,00%	-500,00%	-400,00%	-
8	98,13	12,29	249,78	224,33	0,09	26,99	5,02	194,93	188,29	72,50%	59,15%	21,96%	16,07%
16	5375,35	156,11	> 3h	> 3h	0,51	380,99	170,42	9718,2	6984,62	92,91%	-9,17%	Indef.	Indef.
32	> 3h	1779,07	> 3h	> 3h	7,74	6288,34	1182,62	> 3h	> 3h	Indef.	33,53%	-	-

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias de multiplicadores Dadda foi obtido pelo resolvedor CryptoMiniSAT, um resolvedor SAT apresentado em agosto de 2010 e vencedor do campeonato de resolvedores SATRACE'2010.

Assim como nos testes envolvendo multiplicadores Wallace, em nenhum caso o pré-processador Hiperplic resolveu sozinho o problema de verificação combinatória, mas conseguiu reduzir o tempo de resolução do problema em 8 das 13 instâncias mostradas.

As tabelas 4.5 e 4.6 mostram que, nas pequenas instâncias (de tamanho 1 e 2), o pré-processador e os resolvedores apresentaram tempo de execução muito baixos ou zero, dificultando a análise - estes dados não serão incluídos na análise. Já os resultados obtidos nas instâncias maiores mostram que a aplicação da ferramenta Hiperplic foi vantajosa para os resolvedores SAT utilizados: apenas no *miter* de dois multiplicadores 16x16 o resolvedor CryptoMiniSAT apresentou piora no tempo verificação quando se compara a execução com e sem a utilização da ferramenta Hiperplic (se considerarmos a não utilização das implicações adicionais).

Outra comparação relevante é entre os tempos totais de verificação obtidos quando a geração de implicações extras foi usada (tabela 4.5) e quando não foi usada (tabela 4.6): a utilização das implicações adicionais aumenta o tempo total de verificação e gera um grande impacto negativo no tempo de verificação total, o que permite concluir que a hiper-resolução é suficiente no pré-processamento de multiplicadores Dadda.

4.2.2.3 Multiplicadores Reduced

As tabelas 4.7 e 4.8 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois multiplicadores Reduced. Estes testes foram executados no computador 1, citado na seção 4.1.3.

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias de multiplicadores Reduced foi obtido pelo resolvedor CryptoMi-

Tabela 4.7. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Reduced de tamanhos idênticos. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0,01	0	0	0,03	-	-	-	-	-	-	-	-
2	0	0,01	0	0	0,04	-	-	-	-	-	-	-	-
4	0	0,01	0	0,01	0,06	0,07	0,06	0,06	0,07	-	-500,00%	-	-600,00%
8	59,54	27,16	293,23	217,16	0,84	10,86	19,02	165,2	202,99	81,76%	29,97%	43,66%	6,53%
16	2492,23	3714,21	> 3h	> 3h	43,73	272,79	236,76	6803,89	9165,84	89,05%	93,63%	Indef.	Indef.
32	> 3h	1854,42	> 3h	> 3h	3569,28	13088,5	4696,91	> 3h	> 3h	Indef.	-153,28%	-	-

Tabela 4.8. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores Reduced de tamanhos idênticos. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0,01	0	0	0,03	-	-	-	-	-	-	-	-
2	0	0,01	0	0	0,04	-	-	-	-	-	-	-	-
4	0	0,01	0	0,01	0,05	0,06	0,05	0,05	0,06	-	-400,00%	-	-500,00%
8	59,54	27,16	293,23	217,16	0,09	10,11	18,27	164,45	202,24	83,02%	32,73%	43,92%	6,87%
16	2492,23	3714,21	> 3h	> 3h	0,39	229,45	193,42	6760,55	9122,5	90,79%	94,79%	Indef.	Indef.
32	> 3h	1854,42	> 3h	> 3h	4,77	9524	1132,4	> 3h	> 3h	Indef.	38,94%	-	-

niSAT, assim como nos multiplicadores Wallace e Dadda.

Assim como nos testes envolvendo multiplicadores Wallace e Dadda, em nenhum caso o pré-processador Hiperplic resolveu sozinho o problema de verificação combinatória, mas conseguiu reduzir o tempo de resolução do problema em 10 das 13 instâncias mostradas.

As tabelas 4.7 e 4.8 mostram que, nas pequenas instâncias (de tamanho 1 e 2), o pré-processador e os resolvedores apresentaram tempo de execução muito baixos ou zero, dificultando a análise - estes dados não serão incluídos na análise. Já os resultados obtidos nas instâncias maiores mostram que a aplicação da ferramenta Hiperplic foi vantajosa para os resolvedores SAT utilizados. Se considerados os casos onde apenas a hiper-resolução foi utilizada e o resolvedor de melhor desempenho, a menor redução foi de um terço do tempo original.

Mais uma vez, se comparados os tempos totais de verificação obtidos quando a geração de implicações extras foi usada (tabela 4.7) e quando não foi usada (tabela 4.8), pode se concluir que a utilização das implicações adicionais aumenta o tempo total de verificação. No entanto, ao verificar multiplicadores Reduced, o impacto da utilização de implicações adicionais não é tão ruim: o ganho de performance apenas é reduzido, chegando a ser negativo apenas em um caso.

4.2.2.4 Multiplicadores baseados em somadores *carry save*

As tabelas 4.9 e 4.10 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois multiplicadores baseados em somadores *carry save*. Estes testes foram executados no computador 1, citado na seção 4.1.3.

Tabela 4.9. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores baseados em somadores *carry save* de tamanhos idênticos. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
2	0	0	0	0	0,023	-	-	-	-	-	-	-	-
4	0	0,01	0	0	0,028	0,028	0,038	0,028	0,028	-	-279,96%	-	-
8	7,6	4,77	65,74	50,18	0,05999	4,19999	5,14999	49,45	49,88	44,74%	-7,97%	24,78%	0,60%
16	386,61	> 3h	> 3h	> 3h	0,22997	165,44	> 3h	> 3h	> 3h	57,21%	-	-	-
32	> 3h	-	> 3h	> 3h	1,3378	> 3h	> 3h	> 3h	> 3h	-	-	-	-

Tabela 4.10. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois multiplicadores baseados em somadores *carry save* de tamanhos idênticos. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
2	0	0	0	0	0,02	-	-	-	-	-	-	-	-
4	0	0,01	0	0	0,03	0,03	0,04	0,03	0,03	-	-269,96%	-	-
8	7,6	4,77	65,74	50,18	0,05	4,19	5,14	49,44	49,87	44,92%	-7,67%	24,80%	0,63%
16	386,61	> 3h	> 3h	> 3h	0,12	165,33	> 3h	> 3h	> 3h	57,24%	-	-	-
32	> 3h	-	> 3h	> 3h	0,48	> 3h	> 3h	> 3h	> 3h	-	-	-	-

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias de multiplicadores baseados em somadores *carry save* foi obtido pelo resolvedor CryptoMiniSAT, assim como nos outros multiplicadores testados. Em nenhum caso o pré-processador Hiperplic resolveu sozinho o problema de verificação combinatória e, dentre as instâncias testadas, não houveram melhorias significativas quando comparados os resolvedores SAT com e sem pré-processamento da ferramenta Hiperplic.

Ao contrário dos testes anteriores, a instância de tamanho 1 não foi utilizada por não gerar resultados passíveis de análise. A instância de tamanho 2 também mostrou resultados próximos de zero. As tabelas 4.9 e 4.10 mostram que para instâncias do mesmo tamanho testadas nos multiplicadores de outros tipos, a quantidade de testes de multiplicadores *carry save based* terminados antes do período limite de 3 horas de execução foi bem menor, mesmo quando a ferramenta Hiperplic foi utilizada. De fato, os resultados mostram redução no tempo de execução apenas em instâncias médias de multiplicadores, mas nada significativo.

4.2.3 *Barrel shifters*

Esta seção mostra o resultado dos testes obtidos na verificação de *barrel shifters* de tamanhos variados. As instâncias desses circuitos foram geradas pela ferramenta BenCGen [Andrade et al., 2008a], sendo que o tamanho dos circuitos foi limitado pelos tamanhos que a versão utilizada da BenCGen poderia gerar.

As tabelas 4.11 e 4.12 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois *barrel shifters*. Estes testes foram executados no computador 2, citado na seção 4.1.3.

Tabela 4.11. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois *barrel shifters* de tamanhos idênticos. O pré-processamento aplicado envolveu geração de implicações adicionais e hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0	0	0	0	0,00	0,00	0,00	0,00	-	-	-	-
2	0	0	0	0	0	0,00	0,00	0,00	0,00	-	-	-	-
3	0	0	0	0	0,05	0,05	0,05	0,05	0,05	-	-	-	-
4	0	0,01	0,01	0,01	0,10	0,10	0,10	0,10	0,10	-	-949,83%	-949,83%	-949,83%
5	0,05	0,06	0,04	0,02	0,34	0,34	0,34	0,34	0,34	-579,90%	-466,58%	-749,87%	-1599,74%
6	0,51	0,42	0,24	0,11	1,34	1,34	1,34	1,34	1,34	-163,10%	-219,48%	-459,08%	-1119,82%
7	4,37	3,22	1,74	0,81	5,74	5,74	5,74	5,74	5,74	-31,26%	-78,14%	-229,66%	-608,16%
8	38,67	21,25	30,35	11,31	25,27	25,27	25,27	25,27	25,27	34,64%	-18,94%	16,72%	-123,47%
9	537,6	118,07	534,61	133,98	113,22	113,22	113,22	113,22	113,22	78,94%	4,10%	78,82%	15,49%
10	6713,9	1957,68	> 3h	> 3h	516,55	516,55	516,55	516,55	516,55	92,31%	73,61%	Indef.	Indef.
11	> 3h	> 3h	> 3h	> 3h	2507,19	2507,19	2507,19	2507,19	2507,19	Indef.	Indef.	Indef.	Indef.
12	> 3h	> 3h	> 3h	> 3h	> 3h	> 3h	> 3h	> 3h	> 3h	-	-	-	-

Tabela 4.12. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois *barrel shifters* de tamanhos idênticos. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
1	0	0	0	0	0	0	0	0	0	-	-	-	-
2	0	0	0	0	0	0	0	0	0	-	-	-	-
3	0	0	0	0	0,03	0,03	0,03	0,03	0,03	-	-	-	-
4	0	0,01	0,01	0,01	0,05	0,05	0,05	0,05	0,05	-	-389,93%	-389,93%	-389,93%
5	0,05	0,06	0,04	0,02	0,10	0,10	0,10	0,10	0,10	-97,97%	-64,98%	-147,46%	-394,93%
6	0,51	0,42	0,24	0,11	0,27	0,27	0,27	0,27	0,27	46,28%	34,77%	-14,15%	-149,05%
7	4,37	3,22	1,74	0,81	0,97	0,97	0,97	0,97	0,97	77,76%	69,82%	44,15%	-19,98%
8	38,67	21,25	30,35	11,31	4,71	4,71	4,71	4,71	4,71	87,81%	77,82%	84,47%	58,32%
9	537,6	118,07	534,61	133,98	20,47	20,47	20,47	20,47	20,47	96,19%	82,66%	96,17%	84,72%
10	6713,9	1957,68	> 3h	> 3h	112,59	112,59	112,59	112,59	112,59	98,32%	94,25%	Indef.	Indef.
11	> 3h	> 3h	> 3h	> 3h	871,89	871,89	871,89	871,89	871,89	Indef.	Indef.	Indef.	Indef.
12	> 3h	> 3h	> 3h	> 3h	6347,11	6347,11	6347,11	6347,11	6347,11	Indef.	Indef.	Indef.	Indef.

Analisando os resultados, podemos observar que não há um único resolvedor que apresente consistentemente o melhor desempenho na resolução das instâncias de *barrel shifters*. As instâncias menores (com *shifters* entre 1 e 5 bits - inclusive) não geraram resultados confiáveis para análise, já que o valor absoluto das medidas de tempo é muito próximo do dígito significativo.

Nas instâncias entre 6 e 8 bits o melhor resultado foi obtido pelo resolvedor HaifaSAT quando o pré-processamento baseado em hiper-resolução interno estava ativado. Nas instâncias maiores o melhor resultado foi obtido pelo CryptoMiniSAT, que ainda assim não foi capaz de verificar instâncias maiores que 10 bits.

Em todos os casos de verificação de *barrel shifters*, invariavelmente, a ferramenta Hiperplic foi capaz de encontrar conflitos em implicações no *miter* e terminou o processo de verificação sem a execução de resolvedores SAT.

A tabela 4.11 mostra os tempos de verificação total obtidos com aplicação da hiper-resolução após a geração de implicações adicionais, conseguindo verificar instâncias de 10 bits, se saindo melhor que o melhor resolvedor SAT no tempo limite de três horas.

A tabela 4.12 mostra os tempos de verificação total obtidos com aplicação da hiper-resolução sem a geração de implicações adicionais. Pode-se observar que os tempos obtidos são ainda menores que os mostrados na tabela 4.11, conseguindo verificar todas as instâncias apresentadas dentro do tempo limite e reduzindo em aproximadamente 94% o tempo de verificação do maior *barrel shifter* verificado pelo CryptoMiniSAT.

4.2.4 Circuitos somadores

Esta seção mostra o resultado dos testes obtidos na verificação de somadores de tamanhos variados. As instâncias desses circuitos foram geradas pela ferramenta BenCGen [Andrade et al., 2008a] e o tamanho das instâncias utilizadas nos testes variou de 64 a 4096 bits, dependendo do tipo do somador e do tamanho da instância que BenCGen poderia gerar.

As tabelas 4.13 a 4.22 mostram os resultados obtidos nos testes com instâncias de variados tamanhos de *miters* de dois somadores similares. Estes testes foram executados no computador 2, citado na seção 4.1.3.

Analisando os resultados, podemos observar que o melhor desempenho na resolução das instâncias de somadores de tipos e tamanhos variados foi resolvedor CryptoMiniSAT, assim como nos multiplicadores. Em nenhum caso dos vários testes realizados o pré-processador Hiperplic resolveu sozinho o problema de verificação combinatória e, dentre as instâncias testadas, não houveram melhorias significativas quando comparados os melhores resolvedores SAT com e sem pré-processamento da ferramenta Hiperplic.

O pré-processamento realizado pela ferramenta Hiperplic melhorou apenas o desempenho do resolvedor SAT Berkmin, que, ainda assim, apresentou resultados infe-

Tabela 4.13. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores com *carry lookahead* em bloco. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,12	0,03	0,04	0,04	0,07	0,15	0,11	0,11	0,12	-24,99%	-266,63%	-174,97%	-199,97%
128	0,44	0,07	0,17	0,13	0,12	0,41	0,22	0,26	0,26	7,50%	-209,97%	-51,17%	-97,68%
192	1,07	0,14	0,39	0,28	0,17	0,85	0,35	0,45	0,49	20,75%	-148,55%	-14,87%	-74,28%
256	1,72	0,23	0,58	0,59	0,21	1,54	0,46	0,79	0,88	10,35%	-100,86%	-36,55%	-49,49%
320	2,64	0,36	1,1	1,05	0,26	2,45	0,66	1,19	1,29	7,01%	-84,71%	-8,63%	-23,33%
384	3,92	0,49	1,71	1,44	0,31	3,15	0,82	2,06	1,70	19,57%	-67,95%	-20,64%	-18,26%
448	4,8	0,67	2,36	2,09	0,36	4,52	1,09	2,67	2,66	5,83%	-62,68%	-13,13%	-27,27%
512	6,94	0,81	3,87	3,38	0,41	5,97	1,37	3,23	3,59	13,95%	-69,38%	16,49%	-6,27%
576	8,7	0,97	4,22	5,08	0,47	7,56	1,73	3,92	5,34	13,06%	-78,76%	7,02%	-5,20%
640	12,48	1,26	6,05	5,5	0,52	9,38	2,03	7,09	6,69	24,83%	-61,18%	-17,21%	-21,65%
704	12,66	1,5	8,3	7,77	0,57	11,33	2,29	9,34	9,88	10,52%	-52,53%	-12,50%	-27,13%
768	17,32	1,75	11,14	11,74	0,61	13,76	2,94	12,41	13,03	20,53%	-68,22%	-11,44%	-11,02%
832	19,47	2,15	13,71	12,28	0,68	17,97	3,05	15,67	16,21	7,68%	-42,04%	-14,32%	-32,03%
896	22,79	2,47	19,51	17,17	0,73	20,56	3,64	23,71	19,09	9,76%	-47,57%	-21,55%	-11,21%
960	29,45	2,91	21,95	21,02	0,68	22,99	4,06	30,58	24,59	21,95%	-39,38%	-39,30%	-16,96%
1024	36,33	3,26	30,96	33,19	0,84	28,44	4,81	32,48	21,54	21,71%	-47,66%	-4,92%	35,09%

Tabela 4.14. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores com *carry lookahead* em bloco. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu geração de implicações adicionais e depois aplicação de hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,12	0,03	0,04	0,04	0,07	0,15	0,11	0,11	0,12	-27,49%	-276,63%	-182,47%	-207,47%
128	0,44	0,07	0,17	0,13	0,14	0,43	0,24	0,28	0,28	2,28%	-242,83%	-64,69%	-115,37%
192	1,07	0,14	0,39	0,28	0,20	0,88	0,38	0,48	0,52	18,04%	-169,27%	-22,30%	-84,63%
256	1,72	0,23	0,58	0,59	0,26	1,59	0,51	0,84	0,93	7,39%	-123,03%	-45,34%	-58,13%
320	2,64	0,36	1,1	1,05	0,32	2,51	0,72	1,25	1,35	4,89%	-100,26%	-13,72%	-28,66%
384	3,92	0,49	1,71	1,44	0,38	3,22	0,89	2,13	1,77	17,73%	-82,64%	-24,85%	-23,26%
448	4,8	0,67	2,36	2,09	0,43	4,59	1,16	2,74	2,73	4,46%	-72,53%	-15,93%	-30,43%
512	6,94	0,81	3,87	3,38	0,51	6,07	1,47	3,33	3,69	12,59%	-80,98%	14,06%	-9,05%
576	8,7	0,97	4,22	5,08	0,56	7,65	1,82	4,01	5,43	12,01%	-88,14%	4,86%	-6,99%
640	12,48	1,26	6,05	5,5	0,64	9,50	2,15	7,21	6,81	23,89%	-70,55%	-19,16%	-23,80%
704	12,66	1,5	8,3	7,77	0,70	11,46	2,42	9,47	10,01	9,47%	-61,39%	-14,11%	-28,84%
768	17,32	1,75	11,14	11,74	0,74	13,89	3,07	12,54	13,16	19,78%	-75,71%	-12,61%	-12,14%
832	19,47	2,15	13,71	12,28	0,81	18,10	3,18	15,80	16,34	7,04%	-47,85%	-15,24%	-33,05%
896	22,79	2,47	19,51	17,17	0,90	20,73	3,81	23,88	19,26	9,04%	-54,20%	-22,39%	-12,17%
960	29,45	2,91	21,95	21,02	0,96	23,27	4,34	30,86	24,87	20,98%	-49,24%	-40,61%	-18,33%
1024	36,33	3,26	30,96	33,19	0,96	28,56	4,93	32,60	21,66	21,38%	-51,35%	-5,31%	34,73%

riores aos resultados obtidos por resolvedores recentes como o CryptoMiniSAT sem o pré-processamento de Hiperplic.

O mau resultado apresentado pelo pré-processamento de circuitos somadores, principalmente se comparados com o resolvedor CryptoMiniSAT pode ser explicado pela grande quantidade de portas XOR presentes em circuitos somadores [Katz, 1993]. Enquanto o resolvedor CryptoMiniSAT usa heurísticas refinadas para verificações de implicações CNF extraídas de portas desse tipo [Soos et al., 2009], a hiper-resolução adotada neste trabalho não trabalha bem com portas XOR. A tabela 3.4 mostra a grande quantidade de nodos inseridas no grafo de implicações quando portas XOR são

Tabela 4.15. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *ripple carry*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,05	0,01	0,03	0,03	0,06	0,11	0,08	0,08	0,08	-121,98%	-709,91%	-169,97%	-169,97%
128	0,22	0,04	0,11	0,11	0,10	0,29	0,15	0,16	0,16	-30,90%	-269,96%	-43,62%	-43,62%
192	0,5	0,08	0,27	0,31	0,14	0,66	0,23	0,36	0,36	-31,80%	-186,22%	-32,95%	-15,80%
256	0,87	0,14	0,43	0,53	0,18	1,05	0,32	0,64	0,59	-20,57%	-127,84%	-48,60%	-11,13%
320	1,61	0,25	0,97	0,77	0,22	1,22	0,42	1,33	1,49	24,10%	-68,79%	-37,32%	-93,76%
384	2,99	0,24	1,57	1,17	0,27	1,66	0,56	1,24	1,23	44,58%	-132,07%	21,21%	-4,87%
448	4,18	0,36	2,22	1,84	0,31	3,34	0,73	2,27	2,19	20,05%	-103,32%	-2,34%	-19,13%
512	5,49	0,56	3,21	2,47	0,36	5,45	0,89	2,43	2,40	0,71%	-59,10%	24,27%	2,80%
576	6,23	0,67	3,54	5,76	0,41	6,26	1,07	3,62	3,90	-0,48%	-59,69%	-2,26%	32,29%
640	7,88	0,63	6,52	7,58	0,47	6,72	1,31	5,67	7,21	14,75%	-107,61%	13,07%	4,91%
704	10,59	1,04	8,18	9,49	0,51	8,84	1,55	7,39	7,30	16,51%	-49,22%	9,63%	23,06%
768	11,76	1,15	8,13	8,09	0,57	23,12	1,79	11,98	11,47	-96,59%	-55,56%	-47,34%	-41,77%
832	13,86	1,3	13,53	13,94	0,62	11,85	2,05	11,87	12,24	14,49%	-57,84%	12,25%	12,18%
896	16,85	1,8	12,77	23,58	0,69	18,78	2,34	38,39	37,30	-11,45%	-29,94%	-200,62%	-58,18%
960	23,68	1,73	31,48	38,76	0,75	13,84	2,66	20,54	20,52	41,54%	-53,98%	34,74%	47,05%
1024	25,08	2,22	33,03	39,04	0,81	24,28	2,84	22,48	22,62	3,19%	-27,97%	31,94%	42,06%
1152	35,97	2,84	40,85	29,05	0,95	40,83	3,57	35,15	35,29	-13,51%	-25,63%	13,96%	-21,47%
1280	48,42	3,98	46,06	45,44	1,07	37,68	4,87	45,86	46,99	22,17%	-22,48%	0,42%	-3,42%
1408	59,85	4,67	72,96	107,39	1,22	57,52	5,91	59,17	59,66	3,89%	-26,59%	18,90%	44,44%
1536	66,34	4,5	80,73	128,01	1,35	64,92	7,25	79,41	79,17	2,14%	-61,11%	1,64%	38,15%
1664	96,92	6,59	163,77	121,36	1,52	73,65	8,81	98,20	97,80	24,01%	-33,74%	40,04%	19,41%
1792	131,68	7,48	151,55	184,99	1,68	100,57	10,07	190,49	190,26	23,63%	-34,61%	-25,69%	-2,85%
1920	159,64	8,91	212,8	263,46	1,84	126,68	11,84	182,39	182,92	20,64%	-32,94%	14,29%	30,57%
2048	178,41	10,78	269,15	210,31	2,02	146,83	12,84	199,50	203,84	17,70%	-19,11%	25,88%	3,08%
2176	194,77	11,05	229,81	176,43	2,19	155,44	16,36	467,86	452,99	20,19%	-48,09%	-103,59%	-156,76%
2304	260,89	11,76	315,28	357,93	2,36	148,72	18,65	247,58	247,00	43,00%	-58,58%	21,47%	30,99%
2432	293,89	15,72	384,63	521,99	2,54	308,51	20,70	635,36	632,46	-4,98%	-31,71%	-65,19%	-21,16%
2560	305,9	18,8	281,94	462,74	2,73	305,83	22,36	448,04	451,33	0,02%	-18,93%	-58,91%	2,47%
2688	382,05	18,54	571,26	581,7	2,93	370,55	24,72	483,42	482,36	3,01%	-33,33%	15,38%	17,08%
2816	464,72	19,26	478,73	648,77	3,13	364,41	30,42	736,84	722,78	21,58%	-57,95%	-53,92%	-11,41%
2944	481,03	23,92	532,04	1422,22	2,99	321,50	33,31	560,68	560,66	33,16%	-39,25%	-5,38%	60,58%
3072	527,29	27,24	533,82	661,97	3,49	584,59	34,01	1174,85	1171,53	-10,87%	-24,85%	-120,08%	-76,98%
3200	706,44	33,43	512,1	870,97	3,74	647,16	37,24	855,33	849,07	8,39%	-11,41%	-67,02%	2,51%
3328	779,62	40,87	1274,02	956,08	3,98	762,24	42,72	933,37	932,21	2,23%	-4,52%	26,74%	2,50%
3456	645,21	39,55	806,61	848,68	4,18	583,65	44,96	1302,50	1308,44	9,54%	-13,69%	-61,48%	-54,17%
3584	923,73	42,43	1367,76	972,86	4,40	867,26	50,21	807,04	804,67	6,11%	-18,34%	41,00%	17,29%
3712	991,64	43,79	1029,37	1761,21	4,58	833,81	58,47	1634,27	1624,17	15,92%	-33,52%	-58,76%	7,78%
3840	1362,22	47,28	1065,18	1619,87	4,88	1239,05	59,42	1750,07	1785,80	9,04%	-25,68%	-64,30%	-10,24%
3968	1149,12	56,73	1328,81	1105,62	5,12	778,11	67,44	1323,23	1340,49	32,29%	-18,87%	0,42%	-21,24%
4096	1052,59	59,1	1598,68	1361,8	5,37	1277,16	69,32	2315,49	2226,63	-21,34%	-17,29%	-44,84%	-63,51%

encontradas no circuito, o que leva a aumentar o tamanho do problema.

Tabela 4.16. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *ripple carry*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu geração de implicações adicionais e depois aplicação de hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,05	0,01	0,03	0,03	0,06	0,11	0,08	0,08	0,08	-129,98%	-749,90%	-183,30%	-183,30%
128	0,22	0,04	0,11	0,11	0,11	0,30	0,16	0,17	0,17	-37,26%	-304,96%	-56,35%	-56,35%
192	0,5	0,08	0,27	0,31	0,17	0,69	0,26	0,39	0,39	-37,40%	-221,22%	-43,32%	-24,83%
256	0,87	0,14	0,43	0,53	0,23	1,10	0,37	0,69	0,64	-25,97%	-161,40%	-59,53%	-19,99%
320	1,61	0,25	0,97	0,77	0,30	1,30	0,50	1,41	1,57	19,51%	-98,38%	-44,94%	-103,37%
384	2,99	0,24	1,57	1,17	0,37	1,76	0,66	1,34	1,33	41,11%	-175,39%	14,59%	-13,76%
448	4,18	0,36	2,22	1,84	0,45	3,48	0,87	2,41	2,33	16,72%	-141,93%	-8,60%	-26,68%
512	5,49	0,56	3,21	2,47	0,54	5,63	1,07	2,61	2,58	-2,57%	-91,24%	18,66%	-4,49%
576	6,23	0,67	3,54	5,76	0,63	6,48	1,29	3,84	4,12	-4,06%	-92,97%	-8,56%	28,42%
640	7,88	0,63	6,52	7,58	0,74	6,99	1,58	5,94	7,48	11,33%	-150,30%	8,94%	1,36%
704	10,59	1,04	8,18	9,49	0,85	9,18	1,89	7,73	7,64	13,31%	-81,81%	5,49%	19,49%
768	11,76	1,15	8,13	8,09	0,96	23,51	2,18	12,37	11,86	-99,93%	-89,73%	-52,18%	-46,62%
832	13,86	1,3	13,53	13,94	1,09	12,32	2,52	12,34	12,71	11,11%	-93,83%	8,80%	8,82%
896	16,85	1,8	12,77	23,58	1,21	19,30	2,86	38,91	37,82	-14,53%	-58,77%	-204,68%	-60,38%
960	23,68	1,73	31,48	38,76	1,36	14,45	3,27	21,15	21,13	38,97%	-89,18%	32,81%	45,48%
1024	25,08	2,22	33,03	39,04	1,50	24,97	3,53	23,17	23,31	0,42%	-59,22%	29,84%	40,28%
1152	35,97	2,84	40,85	29,05	1,81	41,69	4,43	36,01	36,15	-15,90%	-55,94%	11,85%	-24,44%
1280	48,42	3,98	46,06	45,44	2,13	38,74	5,93	46,92	48,05	19,99%	-49,04%	-1,87%	-5,75%
1408	59,85	4,67	72,96	107,39	2,52	58,82	7,21	60,47	60,96	1,73%	-54,32%	17,12%	43,24%
1536	66,34	4,5	80,73	128,01	2,95	66,52	8,85	81,01	80,77	-0,26%	-96,57%	-0,34%	36,91%
1664	96,92	6,59	163,77	121,36	3,30	75,43	10,59	99,98	99,58	22,17%	-60,77%	38,95%	17,94%
1792	131,68	7,48	151,55	184,99	3,86	102,75	12,25	192,67	192,44	21,97%	-63,76%	-27,13%	-4,03%
1920	159,64	8,91	212,8	263,46	4,35	129,19	14,35	184,90	185,43	19,08%	-61,01%	13,11%	29,62%
2048	178,41	10,78	269,15	210,31	4,88	149,69	15,70	202,36	206,70	16,10%	-45,66%	24,81%	1,72%
2176	194,77	11,05	229,81	176,43	5,43	158,68	19,60	471,10	456,23	18,53%	-77,41%	-105,00%	-158,59%
2304	260,89	11,76	315,28	357,93	6,02	152,38	22,31	251,24	250,66	41,59%	-89,68%	20,31%	29,97%
2432	293,89	15,72	384,63	521,99	6,62	312,59	24,78	639,44	636,54	-6,36%	-57,65%	-66,25%	-21,95%
2560	305,9	18,8	281,94	462,74	7,29	310,39	26,92	452,60	455,89	-1,47%	-43,19%	-60,53%	1,48%
2688	382,05	18,54	571,26	581,7	7,97	375,59	29,76	488,46	487,40	1,69%	-60,51%	14,49%	16,21%
2816	464,72	19,26	478,73	648,77	8,66	369,94	35,95	742,37	728,31	20,39%	-86,67%	-55,07%	-12,26%
2944	481,03	23,92	532,04	1422,22	9,41	327,92	39,73	567,10	567,08	31,83%	-66,09%	-6,59%	60,13%
3072	527,29	27,24	533,82	661,97	10,40	591,50	40,92	1181,76	1178,44	-12,18%	-50,24%	-121,38%	-78,02%
3200	706,44	33,43	512,1	870,97	10,94	654,36	44,44	862,53	856,27	7,37%	-32,92%	-68,43%	1,69%
3328	779,62	40,87	1274,02	956,08	11,83	770,09	50,57	941,22	940,06	1,22%	-23,74%	26,12%	1,68%
3456	645,21	39,55	806,61	848,68	12,73	592,20	53,51	1311,05	1316,99	8,22%	-35,30%	-62,54%	-55,18%
3584	923,73	42,43	1367,76	972,86	13,58	876,44	59,39	816,22	813,85	5,12%	-39,96%	40,32%	16,35%
3712	991,64	43,79	1029,37	1761,21	14,50	843,73	68,39	1644,19	1634,09	14,92%	-56,17%	-59,73%	7,22%
3840	1362,22	47,28	1065,18	1619,87	15,47	1249,64	70,01	1760,66	1796,39	8,26%	-48,08%	-65,29%	-10,90%
3968	1149,12	56,73	1328,81	1105,62	16,45	789,44	78,77	1334,56	1351,82	31,30%	-38,85%	-0,43%	-22,27%
4096	1052,59	59,1	1598,68	1361,8	17,56	1289,35	81,51	2327,68	2238,82	-22,49%	-37,92%	-45,60%	-64,40%

Tabela 4.17. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry skip*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,08	0,02	0,04	0,03	0,07	0,12	0,09	0,09	0,09	-46,24%	-334,95%	-117,48%	-189,97%
128	0,3	0,05	0,12	0,09	0,11	0,29	0,16	0,19	0,18	4,01%	-215,97%	-56,65%	-97,76%
192	0,76	0,1	0,27	0,2	0,15	0,61	0,26	0,32	0,32	19,74%	-159,98%	-18,51%	-59,99%
256	1,58	0,16	0,58	0,54	0,19	1,07	0,39	0,50	0,53	32,09%	-145,61%	13,28%	1,30%
320	2,27	0,24	0,9	0,87	0,23	1,47	0,57	0,86	0,87	35,33%	-136,65%	4,67%	0,23%
384	3,67	0,34	1,49	1,18	0,28	1,86	0,72	1,29	1,25	49,37%	-111,16%	13,56%	-5,76%
448	4,94	0,45	2,35	1,79	0,32	3,19	0,88	1,97	2,07	35,35%	-96,43%	16,00%	-15,86%
512	6,39	0,52	3,14	2,93	0,37	3,68	1,12	2,70	2,29	42,41%	-115,37%	14,01%	21,84%
576	8,7	0,67	4,61	4,66	0,41	4,72	1,29	4,87	3,21	45,69%	-93,27%	-5,75%	31,01%
640	10,02	0,94	6,27	8,67	0,46	5,71	1,66	4,52	4,60	43,02%	-76,48%	27,93%	46,96%
704	12,7	1,08	8,8	9	0,51	7,03	1,89	9,57	5,43	44,65%	-74,99%	-8,75%	39,67%
768	17,02	1,18	9,13	12,19	0,55	9,33	2,21	8,10	10,79	45,20%	-87,03%	11,32%	11,51%
832	20,62	1,48	11,22	10,69	0,59	10,86	2,54	10,38	9,38	47,32%	-71,75%	7,47%	12,24%
896	24,45	1,8	14,8	14,38	0,62	14,05	2,95	14,73	10,64	42,52%	-64,16%	0,44%	25,97%
960	28,3	2,06	17,88	22,62	0,71	15,91	3,25	12,07	16,43	43,77%	-57,86%	32,48%	27,36%
1024	34,57	2,23	19,9	24,2	0,72	20,08	3,72	17,08	20,63	41,91%	-66,95%	14,16%	14,74%

Tabela 4.18. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry skip*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu geração de implicações adicionais e depois aplicação de hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,08	0,02	0,04	0,03	0,07	0,12	0,09	0,09	0,09	-51,24%	-354,95%	-127,48%	-203,30%
128	0,3	0,05	0,12	0,09	0,12	0,30	0,17	0,20	0,19	1,01%	-233,97%	-64,15%	-107,76%
192	0,76	0,1	0,27	0,2	0,16	0,62	0,27	0,33	0,33	17,90%	-173,97%	-23,69%	-66,99%
256	1,58	0,16	0,58	0,54	0,21	1,09	0,41	0,52	0,55	30,95%	-156,86%	10,18%	-2,03%
320	2,27	0,24	0,9	0,87	0,25	1,49	0,59	0,88	0,89	34,41%	-145,40%	2,34%	-2,18%
384	3,67	0,34	1,49	1,18	0,30	1,88	0,74	1,31	1,27	48,75%	-117,93%	12,02%	-7,71%
448	4,94	0,45	2,35	1,79	0,35	3,22	0,91	2,00	2,10	34,82%	-102,21%	14,90%	-17,32%
512	6,39	0,52	3,14	2,93	0,40	3,71	1,15	2,73	2,32	41,86%	-122,10%	12,90%	20,65%
576	8,7	0,67	4,61	4,66	0,45	4,76	1,33	4,91	3,25	45,28%	-98,65%	-6,53%	30,24%
640	10,02	0,94	6,27	8,67	0,51	5,76	1,71	4,57	4,65	42,51%	-82,01%	27,10%	46,36%
704	12,7	1,08	8,8	9	0,57	7,09	1,95	9,63	5,49	44,19%	-80,36%	-9,41%	39,02%
768	17,02	1,18	9,13	12,19	0,60	9,38	2,26	8,15	10,84	44,87%	-91,86%	10,69%	11,04%
832	20,62	1,48	11,22	10,69	0,65	10,92	2,60	10,44	9,44	47,04%	-75,74%	6,94%	11,68%
896	24,45	1,8	14,8	14,38	0,70	14,13	3,03	14,81	10,72	42,20%	-68,44%	-0,08%	25,44%
960	28,3	2,06	17,88	22,62	0,76	15,96	3,30	12,12	16,48	43,60%	-60,29%	32,20%	27,14%
1024	34,57	2,23	19,9	24,2	0,79	20,15	3,79	17,15	20,70	41,71%	-69,99%	13,81%	14,46%

Tabela 4.19. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry save*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,07	0,01	0,03	0,02	0,06	0,09	0,08	0,08	0,08	-31,42%	-719,91%	-173,30%	-309,96%
128	0,61	0,04	0,15	0,07	0,10	0,23	0,17	0,18	0,18	61,97%	-329,96%	-21,32%	-159,98%
192	1,45	0,09	0,53	0,19	0,14	0,48	0,28	0,33	0,33	66,76%	-213,31%	37,36%	-74,73%
256	-	0,13	1,25	0,4	0,18	0,88	0,42	0,60	0,60	-	-224,59%	51,84%	-50,49%
320	3,45	0,22	2,48	1,08	0,22	1,45	0,58	0,89	0,90	57,97%	-163,62%	64,11%	16,67%
384	5,34	0,33	3,49	1,09	0,27	2,15	0,81	1,58	1,90	59,81%	-144,23%	54,84%	-73,94%
448	9,04	0,45	5,03	1,8	0,31	3,11	1,05	2,93	2,98	65,64%	-132,43%	41,83%	-65,33%
512	7,3	0,51	8	2,31	0,35	4,36	1,25	4,10	4,04	40,26%	-145,28%	48,74%	-74,93%
576	10,19	0,64	12,92	3,29	0,37	7,75	1,55	5,42	5,46	23,91%	-142,80%	58,02%	-66,08%
640	17,96	0,77	16,48	4,87	0,43	7,59	1,90	4,98	5,01	57,76%	-146,36%	69,80%	-2,81%
704	18,68	1,01	20,42	5,75	0,48	9,63	2,36	8,76	8,74	48,43%	-133,95%	57,09%	-52,05%
768	16,56	1,46	24,25	8,35	0,52	12,31	2,58	9,22	9,28	25,66%	-76,71%	61,98%	-11,14%
832	18,99	1,39	36,08	14,3	0,57	15,31	3,00	17,95	17,58	19,40%	-115,53%	50,26%	-22,91%
896	26,4	1,79	42,75	11,54	0,61	19,00	3,59	14,52	14,64	28,01%	-100,83%	66,02%	-26,91%
960	35,91	3,28	98,06	13,44	0,65	22,88	4,18	16,46	17,62	36,28%	-27,53%	83,21%	-31,12%
1024	34,05	3	92,54	15,04	0,70	27,71	4,68	34,42	34,31	18,61%	-56,16%	62,80%	-128,16%

Tabela 4.20. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry save*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu geração de implicações adicionais e depois aplicação de hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,07	0,01	0,03	0,02	0,06	0,09	0,08	0,08	0,08	-32,84%	-729,90%	-176,63%	-314,95%
128	0,61	0,04	0,15	0,07	0,11	0,24	0,18	0,19	0,19	60,66%	-349,96%	-26,66%	-171,40%
192	1,45	0,09	0,53	0,19	0,15	0,49	0,29	0,34	0,34	65,86%	-227,75%	34,91%	-81,57%
256	-	0,13	1,25	0,4	0,20	0,90	0,44	0,62	0,62	-	-236,13%	50,64%	-54,24%
320	3,45	0,22	2,48	1,08	0,24	1,47	0,60	0,91	0,92	57,28%	-174,53%	63,15%	14,45%
384	5,34	0,33	3,49	1,09	0,29	2,17	0,83	1,60	1,92	59,40%	-150,90%	54,21%	-75,96%
448	9,04	0,45	5,03	1,8	0,34	3,14	1,08	2,96	3,01	65,29%	-139,54%	41,19%	-67,11%
512	7,3	0,51	8	2,31	0,38	4,39	1,28	4,13	4,07	39,85%	-151,17%	48,36%	-76,23%
576	10,19	0,64	12,92	3,29	0,43	7,81	1,61	5,48	5,52	23,37%	-151,40%	57,59%	-67,75%
640	17,96	0,77	16,48	4,87	0,45	7,61	1,92	5,00	5,03	57,64%	-149,08%	69,67%	-3,24%
704	18,68	1,01	20,42	5,75	0,52	9,67	2,40	8,80	8,78	48,22%	-137,81%	56,90%	-52,73%
768	16,56	1,46	24,25	8,35	0,57	12,36	2,63	9,27	9,33	25,37%	-80,06%	61,78%	-11,72%
832	18,99	1,39	36,08	14,3	0,62	15,36	3,05	18,00	17,63	19,09%	-119,71%	50,10%	-23,31%
896	26,4	1,79	42,75	11,54	0,64	19,03	3,62	14,55	14,67	27,93%	-102,06%	65,97%	-27,10%
960	35,91	3,28	98,06	13,44	0,72	22,95	4,25	16,53	17,69	36,10%	-29,51%	83,15%	-31,61%
1024	34,05	3	92,54	15,04	0,76	27,77	4,74	34,48	34,37	18,43%	-58,13%	62,74%	-128,55%

Tabela 4.21. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry select*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu apenas hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,25	0,06	0,12	0,09	0,12	0,29	0,17	0,21	0,22	-15,59%	-181,64%	-74,15%	-143,31%
128	1,04	0,18	0,38	0,31	0,22	0,90	0,38	0,51	0,52	12,98%	-113,87%	-35,52%	-69,34%
192	2,09	0,37	0,99	0,7	0,33	1,75	0,65	1,10	1,03	16,46%	-74,58%	-10,70%	-46,56%
256	3,81	0,59	1,53	1,42	0,43	3,24	0,93	2,14	1,86	14,94%	-57,79%	-39,93%	-31,05%
320	5,62	0,87	3,35	2,57	0,55	4,57	1,37	3,09	2,87	18,74%	-57,12%	7,85%	-11,55%
384	9,21	1,28	5,16	4,73	0,66	7,01	1,77	4,71	4,38	23,87%	-38,43%	8,68%	7,36%
448	13,45	1,68	8,81	7,87	0,77	9,66	2,26	6,49	6,99	28,19%	-34,46%	26,35%	11,20%
512	17,65	2,17	13,12	11,41	0,90	12,64	2,92	9,13	10,20	28,36%	-34,79%	30,37%	10,56%
576	23,67	2,72	18,33	17,77	1,00	17,36	3,45	11,87	13,32	26,66%	-26,80%	35,25%	25,05%
640	30,06	3,61	22,5	21,7	1,12	22,00	4,36	16,76	16,86	26,82%	-20,74%	25,52%	22,31%
704	35,1	4,16	27,94	29,5	1,23	27,23	5,17	20,74	20,92	22,42%	-24,27%	25,77%	29,09%
768	46,06	5,02	35,54	32,2	1,34	32,21	5,91	23,16	24,61	30,06%	-17,80%	34,82%	23,56%
832	50,22	5,89	46,94	41,22	1,32	42,12	7,05	32,51	31,44	16,13%	-19,67%	30,74%	23,73%
896	62,4	6,85	50,52	49,05	1,55	48,42	8,12	47,08	34,62	22,40%	-18,61%	6,80%	29,41%
960	79,68	7,89	60,08	56,78	1,69	54,22	9,70	42,02	45,62	31,96%	-22,90%	30,07%	19,66%
1024	97,91	8,97	75,79	72,8	1,81	68,45	11,56	51,44	47,90	30,09%	-28,84%	32,13%	34,21%

Tabela 4.22. Desempenho obtido na verificação de circuitos *miter* criados a partir de dois somadores *carry select*. Ambos os somadores tinham tamanho idêntico. O pré-processamento aplicado envolveu geração de implicações adicionais e depois aplicação de hiper-resolução.

T.	Tempo referência (s)				T.	Tempo total verif. (s)				Ganho			
c.	BM	CMS	HF	HFP	pré	BM	CMS	HF	HFP	BM	CMS	HF	HFP
64	0,25	0,06	0,12	0,09	0,15	0,32	0,20	0,24	0,25	-26,39%	-226,63%	-96,65%	-173,31%
128	1,04	0,18	0,38	0,31	0,31	0,99	0,47	0,60	0,61	4,33%	-163,86%	-59,20%	-98,37%
192	2,09	0,37	0,99	0,7	0,52	1,94	0,84	1,29	1,22	7,32%	-126,19%	-29,99%	-73,85%
256	3,81	0,59	1,53	1,42	0,74	3,55	1,24	2,45	2,17	6,75%	-110,66%	-60,32%	-53,02%
320	5,62	0,87	3,35	2,57	1,01	5,03	1,83	3,55	3,33	10,52%	-110,21%	-5,94%	-29,53%
384	9,21	1,28	5,16	4,73	1,33	7,68	2,44	5,38	5,05	16,64%	-90,45%	-4,22%	-6,72%
448	13,45	1,68	8,81	7,87	1,66	10,55	3,15	7,38	7,88	21,53%	-87,72%	16,19%	-0,17%
512	17,65	2,17	13,12	11,41	2,04	13,78	4,06	10,27	11,34	21,91%	-87,22%	21,70%	0,59%
576	23,67	2,72	18,33	17,77	2,45	18,81	4,90	13,32	14,77	20,54%	-80,10%	27,34%	16,89%
640	30,06	3,61	22,5	21,7	2,89	23,77	6,13	18,53	18,63	20,92%	-69,88%	17,63%	14,14%
704	35,1	4,16	27,94	29,5	3,38	29,38	7,32	22,89	23,07	16,29%	-76,05%	18,06%	21,78%
768	46,06	5,02	35,54	32,2	3,89	34,76	8,46	25,71	27,16	24,53%	-68,51%	27,66%	15,65%
832	50,22	5,89	46,94	41,22	4,46	45,26	10,19	35,65	34,58	9,88%	-72,99%	24,05%	16,11%
896	62,4	6,85	50,52	49,05	4,93	51,80	11,50	50,46	38,00	16,98%	-67,95%	0,11%	22,52%
960	79,68	7,89	60,08	56,78	5,56	58,09	13,57	45,89	49,49	27,10%	-71,95%	23,62%	12,84%
1024	97,91	8,97	75,79	72,8	6,30	72,94	16,05	55,93	52,39	25,50%	-78,92%	26,21%	28,04%

Capítulo 5

Conclusão

O presente trabalho apresentou uma ferramenta capaz de auxiliar o processo de verificação de equivalência combinatória. Esta ferramenta trabalha como um pré-processador de resolvedores SAT e permite que sejam executadas verificações em menor tempo se comparado com o mesmo resolvedor SAT trabalhando de forma independente. A ferramenta cria um grafo de implicações simples, diretas e indiretas a partir da especificação de um circuito no formato BENCH e aplica uma série de simplificações sobre o grafo, reduzindo significativamente a instância SAT a ser resolvida.

Inicialmente é construído um grafo com implicações simples obtidas através das próprias portas lógicas do circuito, depois são acrescentadas as implicações diretas e indiretas e então é iniciada a etapa de simplificação, que consiste basicamente da propagação de sinais unitários, eliminação de sinais equivalentes e a aplicação da hiper-resolução binária. O processo de simplificação é realizado iterativamente até que nenhuma redução no grafo de implicações possa ser realizada, quando então é gerada uma expressão na forma normal conjuntiva para aplicação em um resolvedor SAT. Durante as etapas de simplificação o pré-processador pode encontrar conflitos ou determinar a satisfabilidade do problema, o que pode dispensar a execução do resolvedor em si.

Nos testes realizados, a aplicação da ferramenta mostrou redução satisfatória na verificação de circuitos multiplicadores e de *barrel shifters*. No primeiro grupo de circuitos houveram picos de reduções de até 94% do tempo total de verificação. No segundo grupo a ferramenta implementada no trabalho conseguiu atingir performance semelhante, com a vantagem de permitir a verificação de instâncias maiores do mesmo problema e sem a necessidade de execução de qualquer resolvedor SAT: a própria ferramenta de pré-processamento foi capaz de executar a verificação. A verificação de circuitos somadores, no entanto, se mostrou ineficaz quando o pré-processamento feito pela ferramenta foi aplicado.

Apesar das grandes reduções proporcionadas principalmente pela eliminação de sinais equivalentes e pela hiper-resolução binária proporcionarem sensível aceleração no processo total de verificação de equivalência combinatória e permitirem a verificação de circuitos maiores, estas mesmas operações ignoram a informação de estrutura do circuito, dificultando a verificação de erros em pontos específicos do mesmo. Na prática, a aplicação do processo de hiper-resolução e outras simplificações têm o mesmo efeito da conversão do circuito para uma fórmula CNF: a perda de informação estrutural.

Como trabalho futuro, recomenda-se elaborar um método de "blindagem" das implicações relativas às portas lógicas, para que evitar que as implicações de uma mesma porta não sejam desmembradas durante a simplificação e assim permitir a reconstrução do circuito para eventuais análises de erros e/ou conflitos.

Referências Bibliográficas

Akers, S. B. (1978). Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516.

Andrade, F. V. (2008). *Contribuições para o problema de verificação de equivalência combinacional*. PhD thesis, Departamento de Ciência da Computação da Universidade Federal de Minas Gerais.

Andrade, F. V.; Oliveira, M. C. M.; Fernandes, A. O. & Júnior, C. N. C. (2007). SAT-based equivalence checking based on circuit partitioning and special approaches for conlict clause reuse. *IEEE, Design and Diagnostics of Electronic Circuits and Systems*, pp. 1–6.

Andrade, F. V.; Silva, L. M. & Fernandes, A. O. (2008a). Bencgen: a digital circuit generation tool for benchmarks. In *SBCCI '08: Proceedings of the 21st annual symposium on Integrated circuits and system design*, pp. 164–169, New York, NY, USA. ACM.

Andrade, F. V.; Silva, L. M. & Fernandes, A. O. (2008b). SAT-based combinational equivalence checking through circuit preprocessing. *Proceedings of IEEE International Conference on Computer Design*, pp. 40–45.

Arora, R. & Hsiao, M. (2004). Using global structural relationships of signals to accelerate SAT-based combinational equivalence checking. 10(12):1597–1628.

Arora, R. & Hsiao, M. S. (2003). Enhancing SAT-based equivalence checking with static logic implications. In *HLDVT'03: Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop*, p. 63, Washington, DC, USA. IEEE Computer Society.

Bacchus, F. & Winter, J. (2003). Effective preprocessing with hyper-resolution and equality reduction. In *In SAT*, pp. 341–355.

Barth, P. & Stadtwald, I. (1995). A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization.

Bayardo, R. J. & Schrag, R. C. (1997). Using CSP look-back techniques to solve real-world SAT instances. pp. 203--208. AAAI Press.

Becker, B.; Drechsler, R. & Enders, R. (1997). On the representational power of bit-level and word-level decision diagrams. In *In Proceedings of the Asia and South Pacific-Design Automation Conference (ASP-DAC)*, pp. 461–467.

Beizer, B. (1995). The pentium bug, an industry watershed. *Testing Techniques Newsletter*.

Bentley, B.; Baty, K.; Normoyle, K.; Ishii, M. & Yogev, E. (2004). Verification: what works and what doesn't. *Proceeding of ACM/IEEE Conference on Design Automation*, p. 274.

Bhattacharya, D. & Hayes, J. P. (1989). *Hierarchical Modeling for VLSI Circuit Testing*. Springer.

Biere, A. & Kunz, W. (2002). SAT and ATPG: Boolean engines for formal hardware verification. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pp. 782–785, New York, NY, USA. ACM.

Bollig, B. & Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. *Computers, IEEE Transactions on*, 45(9):993 –1002.

Brand, D. (1993). Verification of large synthesized designs. *IEEE/ACM International Conference on Computer-Aided Design*, pp. 534–537.

Brglez, F. & Fujiwara, H. (1985). A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. *Proceedings of the International Symposium on Circuits and Systems*, pp. 663–698.

Bryant, R. (1986). Graph-based algorithms for boolean function manipulations. *IEEE Transactions on Computers*, C-35(8):677–691.

Bryant, R. E. (1998). On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40:205–213.

Bryant, R. E. & Chen, Y.-A. (1995). Verification of arithmetic circuits with binary moment diagrams. In *DAC'95: Proceedings of the 32nd ACM/IEEE conference on Design automation*, pp. 535–541, New York, NY, USA. ACM.

Chen, J.-C. & Chen, Y.-A. (2001). Equivalence checking of integer multipliers. *Proceedings of the Asia South Pacific Design Automation Conference*, pp. 169–174.

Clarke, E. & Zhao, X. (1995). Word level symbolic model checking: A new approach for verifying arithmetic circuits. Technical report, Pittsburgh, PA, USA.

Clarke, E. M.; Fujita, M. & Zhao, X. (1995). Hybrid decision diagrams - overcoming the limitations of MTBDDs and BMDs. In *In Int'l Conf. on CAD*, pp. 159–163. IEEE Computer Society Press.

Clarke, E. M.; Grumberg, O. & Peled, D. A. (1999). *Model checking*. MIT Press.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *STOC'71: Proceedings of the third annual ACM symposium on Theory of computing*, pp. 151–158, New York, NY, USA. ACM.

Coudert, O. (1996). On solving covering problems. In *DAC'96: Proceedings of the 33rd annual Design Automation Conference*, pp. 197--202, New York, NY, USA. ACM.

Davis, M.; Logemann, G. & Loveland, D. (1962). A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397.

Davis, M. & Putnam, H. (1960). A computing procedure for quantification theory. *J. ACM*, 7(3):201--215.

Drechsler, R. (2004). *Advanced Formal Verification*. Springer, 1st edição.

Drechsler, R.; Becker, B. & Ruppertz, S. (1997). The K*BMD: A verification data structure. *IEEE Des. Test*, 14(2):51–59.

Eén, N. & Sörensson, N. (2003). An extensible SAT-solver. *International Conference on Theory and Applications of Satisfability Testing*, pp. 840–843.

Ferrandi, F.; Macii, A.; Macii, E.; Poncino, M.; Scarsi, R. & Somenzi, F. (1998). Symbolic algorithms for layout-oriented synthesis of pass transistor logic circuits. In *ICCAD'98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pp. 235--241, New York, NY, USA. ACM.

Flores, P. F.; Neto, H. C. & Silva, J. P. M. (2001). An exact solution to the minimum size test pattern problem. *ACM Trans. Des. Autom. Electron. Syst.*, 6(4):629--644.

Foster, H.; Krolnik, A. C. & Lacey, D. J. (2004). *Assertion-based Design.* Springer, 2nd edição.

Fujita, M.; Fujisawa, H. & Kawato, N. (1988). Evaluation and improvement of boolean comparison method based on binary decision diagrams. In *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on*, pp. 2 –5.

Fujiwara, H. & Shimono, T. (1983). On the acceleration of test generation algorithms. *IEEE Trans. Comput.*, 32(12):1137–1144.

Gershman, R. (2005). HaifaSat: a new robust SAT solver. In *Proceedings of the 1st International Haifa Verification Conference, LNCS 3875*, pp. 76--89.

Gershman, R. & Strichman, O. (2005). Cost-effective hyper-resolution for preprocessing CNF formulas. In *Theory and Applications of Satisfiability Testing (SAT05)*, pp. 423–429.

Gizdarski, E. & Fujiwara, H. (2001). SPIRIT: A highly robust combinational test generation algorithm. In *19 th IEEE Proc. on VTS, 2001*, pp. 346–351.

Goldberg, E. & Novikov, Y. (2002). BerkMin: A fast and robust sat-solver. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, p. 142, Washington, DC, USA. IEEE Computer Society.

Goldberg, E.; Prasad, M. & Brayton, R. (2001). Using SAT for combinational equivalence checking. In *Proceedings of the conference on Design, automation and test in Europe*, pp. 114–121, Piscataway, NJ, USA. IEEE Press.

Gupta, A. & Ashar, P. (1998). Integrating a boolean satisfiability checker and BDDs for combinational equivalence checking. In *VLSID '98: Proceedings of the Eleventh International Conference on VLSI Design: VLSI for Signal Processing*, p. 222, Washington, DC, USA. IEEE Computer Society.

Hamaguchi, K.; Morita, A. & Yajima, S. (1995). Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pp. 78–82.

Hatchel, G. D. & Somenzi, F. (2000). *Logic Synthesis and Verification Algorithms.* Kluwer Academic Publishers.

Ibarra, O. & Sahni, S. (1975). Polynomially complete fault detection problems. *IEEE Transactions on Computers*, 24:242–249.

Katz, R. H. (1993). *Contemporary Logic Design.* Prentice Hall.

Keim, M.; Drechsler, R.; Becker, B.; Martin, M. & Molitor, P. (2003). Polynomial formal verification of multipliers. *Form. Methods Syst. Des.*, 22(1):39–58.

Kirkland, T. & Mercer, M. R. (1987). A topological search algorithm for ATPG. In *DAC'87: Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 502–508, New York, NY, USA. ACM.

Kunz, W. (1993). HANNIBAL: an efficient tool for logic verification based on recursive learning. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pp. 538–543, Los Alamitos, CA, USA. IEEE Computer Society Press.

Kunz, W. & Pradhan, D. (1993). Accelerated dynamic learning for test pattern generation in combinational circuits. *IEEE Transactions on Computer Aided Design*, 12(5).

Kunz, W. & Pradhan, D. K. (1994). Recursive learning: A new implication technique for efficient solutions to cad-problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1143–1158.

Larrabee, T. (1990). *Efficient generation of test patterns using Boolean satisfiability.* PhD thesis, Stanford, CA, USA.

Larrabee, T. (1992). Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 11:4--15.

Lee, C. Y. (1959). Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal.*

Lee, H. K. & Ha, D. S. (1990). SOPRANO: an efficient automatic test pattern generator for stuck-open faults in CMOS combinational circuits. In *DAC'90: Proceedings of the 27th ACM/IEEE Design Automation Conference*, pp. 660–666, New York, NY, USA. ACM.

Lu, F.; Wang, L.-C.; Moondanos, J. & Hanna, Z. (2004). A signal correlation guided circuit-SAT solver. 10(12):1629--1654.

Luk, W. K. & Vuillemin, J. E. (1983). Recursive implementation of optimal time VLSI integer multipliers. *VLSI'83: VLSI design of digital systems*.

Malik, S.; Wang, A.; Brayton, R. & Sangiovanni-Vincentelli, A. (1988). Logic verification using binary decision diagrams in a logic synthesis environment. In *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on*, pp. 6 –9.

Mendes, A. J. (2008). Verificação de equivalência de circuitos combinacionais dissimilares através do reaproveitamento de cláusulas de conflito. Master's thesis, Departamento de Ciência da Computação da Universidade Federal de Minas Gerais.

Mishchenko, A.; Chatterjee, S.; Brayton, R. & Een, N. (2006). Improvements to combinational equivalence checking. *IEEE/ACM International Conference on Computer-Aided Design*, pp. 836–843.

Mohammadi, M.; Pazhoumand-dar, H.; Soryani, M. & Moeinzadeh, H. (2009). HS-ROBDD: an efficient variable order binary decision diagram. In *GECCO'09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*, pp. 2115–2118, New York, NY, USA. ACM.

Molitor, P. & Mohnke, J. (2004). *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Springer.

Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics Magazine*, 38(8):114–117.

Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; Zhang, L. & Malik, S. (2001). Chaff: engineering an efficient SAT solver. In *DAC '01: Proceedings of the 38th annual Design Automation Conference*, pp. 530--535, New York, NY, USA. ACM.

Murgai, R.; Jain, J. & Fujita, M. (1999). Efficient scheduling techniques for ROBDD construction. In *VLSID'99: Proceedings of the 12th International Conference on VLSI Design - 'VLSI for the Information Appliance'*, p. 394, Washington, DC, USA. IEEE Computer Society.

Oliveira, M. C. M. (2006). Um núcleo inteligente para processamento distribuído de resolvedores SAT em verificação por equivalência. Master's thesis, Departamento de Ciência da Computação da Universidade Federal de Minas Gerais.

Prasad, M. R.; Biere, A. & Gupta, A. (2005). A survey of recent advances in SAT-based formal verification. In *International Journal on Software Tools for Technology Transfer (STTT)*, volume 7, pp. 156–173. Springer Berlin/Heidelberg.

Robinson, J. A. (1965). Automatic deduction with hyper-resolution. *International Journal on Computer Math*, pp. 227–234.

Roth, J. P. (1966). Diagnosis of automata failures: a calculus and a method. *IBM J. Res. Dev.*, 10(4):278–291.

Schaller, R. R. (1997). Moore's law: past, present and future. *IEEE Spectrum*, pp. 53–59.

Scholl, C.; Becker, B. & Brogle, A. (2001). The multiple variable order problem for binary decision diagrams: theory and practical application. In *ASP-DAC '01: Proceedings of the 2001 Asia and South Pacific Design Automation Conference*, pp. 85–90, New York, NY, USA. ACM.

Schulz, M. H.; Trischler, E. & Sarfert, T. M. (1988). SOCRATES: A highly efficient automatic test pattern generation system. *IEEE transactions on computer-aided design of integrated circuits and systems*, 7(1):129–137.

Silva, J. M. (2000). Algebraic simplification techniques for propositional satisfiability. Lecture Notes in Computer Science.

Silva, J. M. & Glass, T. (1999). Combinational equivalence checking using satisfiability and recursive learning. In *DATE'99: Proceedings of the conference on Design, automation and test in Europe*, p. 33, New York, NY, USA. ACM.

Silva, J. M. & Silva, L. G. e. (1999). Algorithms for satisfiability in combinational circuits based on backtrack search and recursive learning. *Workshop Notes of the International Workshop on Logic Synthesis*, pp. 227–241.

Silva, J. P. M. & Sakallah, K. A. (1996). GRASP - a new search algorithm for satisfiability. In *ICCAD '96: Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design*, pp. 220–227, Washington, DC, USA. IEEE Computer Society.

Smith, D. J. (1996). VHDL & Verilog compared & contrasted–plus modeled example written in VHDL, verilog and c. In *DAC '96: Proceedings of the 33rd annual Design Automation Conference*, pp. 771–776, New York, NY, USA. ACM.

Soos, M.; Nohl, K. & Castelluccia, C. (2009). Extending SAT solvers to cryptographic problems. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*, SAT '09, pp. 244--257, Berlin, Heidelberg. Springer-Verlag.

Subbarayan, S. & Pradhan, D. K. (2004). NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. In *In Proc. 7th International Conference on Theory and Applications of Satisfiability Testing (SAT*, pp. 276--291. Springer.

Tafertshofer, P.; Ganz, A. & Antreich, K. (2000). IGRAINE-an Implication GRaph-bAsed engINE for fast implication, justification, and propagation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(8):907 –927.

Tanenbaum, A. S. (2001). *Organização Estruturada de Computadores*. LTC, 4th edição.

Tarjan, R. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160.

Van Der Schoot, H. & Ural, H. (1996). A uniform approach to tackle state explosion in verifying progress properties for networks of CFSMs.

van Eijk, C. & Janssen, G. L. J. M. (1994). Exploiting structural similarities in a BDD-based verification method. In *in Theorem Provers in Circuit Design. 1994, number 901 in Lecture Notes in Computer Science*, pp. 110–125. Springer-Verlag.

Wallace, C. S. (1964). A suggestion for a fast multiplier. *IEEE transactions on electronic computers*.

Wang, L.-T.; Chang, Y.-W. & Cheng, K.-T. (2009). *Electronic Design Automation*. Morgan Kaufmann.

Wefel, S. & Molitor, P. (2000). Prove that a faulty multiplier is faulty!? In *GLSVLSI '00: Proceedings of the 10th Great Lakes symposium on VLSI*, pp. 43–46, New York, NY, USA. ACM.

Xu, Z.; Yan, X.; Lu, Y. & Ge, H. (2003). Equivalence checking using independent cuts. *12th Asian Test Symposium*, pp. 482–485.

Zhang, H. (1997). SATO: An efficient propositional prover. In *CADE-14: Proceedings of the 14th International Conference on Automated Deduction*, pp. 272--275, London, UK. Springer-Verlag.

Zhao, J.-K.; Rudnick, E. & Patel, J. (1997). Static logic implication with application to redundancy identification. In *VLSI Test Symposium, 1997., 15th IEEE*, pp. 288 –293.

Apêndice A

Formato de arquivo BENCH

O formato BENCH é um arquivo de texto convenientemente utilizado para representar circuitos aritméticos, contendo descrições de portas lógicas associadas a variáveis. As portas lógicas válidas em um circuito combinacional no formato BENCH são as seguintes:

- AND
- NAND
- OR
- NOR
- BUF
- NOT
- XOR
- XNOR

Com exceção de BUFF e NOT que aceitam somente uma variável como entrada, as outras portas aceitam como entrada duas ou mais variáveis.

Além de aceitar a especificação de portas lógicas, o formato permite a definição das entradas (INPUT) e saídas (OUTPUT) do circuito. As variáveis são nomeadas por combinações de caracteres alfanuméricos, desde que não se inicie com *underline* (_) ou contenha caracteres separadores: “,”, “(”, “)” e “=”. Uma variável também não pode aparecer mais de uma vez do lado esquerdo de uma igualdade, pois isso representaria um mesmo sinal sendo controlado por saídas de duas portas lógicas diferentes. O

Listagem A.1. Exemplo de arquivo no formato BENCH para o circuito da figura 3.2.

```
# Sinais de entrada
INPUT(1)
INPUT(2)
INPUT(3)

# Sinais de saída
OUTPUT(12)
# Descrição do circuito
4 = NOT(1)
5 = NOT(2)
6 = BUF(3)
7 = AND(2,3)
8 = OR(5,6)
9 = AND(1,7)
10 = OR(4,8)
11 = NOT(9)
12 = XOR(10,11)
```

caractere "#" inicia um comentário, que termina ao fim da linha. O arquivo mostrado na listagem A.1 corresponde ao circuito da figura 3.2 e o arquivo mostrado na listagem A.2 corresponde ao circuito da figura 3.10.

Listagem A.2. Exemplo de arquivo no formato BENCH para o circuito da figura 3.10.

```
# Sinais de entrada
INPUT(1)
INPUT(2)
INPUT(3)

# Sinais de saída
OUTPUT(12)
# Descrição do circuito
4 = NOT(1)
5 = NOT(2)
6 = NOT(3)
7 = AND(2,3)
8 = OR(5,6)
9 = AND(1,7)
10 = OR(4,8)
11 = NOT(9)
12 = XOR(10,11)
```

Apêndice B

Formato de arquivo DIMACS

O formato DIMACS CNF é o formato de arquivo mais comum para armazenar uma expressão booleana na forma normal conjuntiva. O início do arquivo se dá com linhas de comentários, iniciadas pela letra "c" ou por um cabeçalho no formato "p cnf *variáveis cláusulas*", onde *variáveis* informa o número de variáveis na expressão booleana e *cláusulas* informa o número de cláusulas.

Cada variável é representada por um valor inteiro não nulo, positivo para representar um literal normal (positivo), ou negativo para representar um literal barrado (negativo). As variáveis não precisam ser declaradas, sendo que o único requisito é que seu valor não seja maior em módulo que o número de variáveis declarado no cabeçalho.

As cláusulas são definidas uma por linha, mostrando apenas as variáveis positivas ou negativas presentes, e terminadas com o valor "0" (sem as aspas).

$$\begin{aligned} \gamma \;=\; & (\overline{\sigma_1} \vee \sigma_2 \vee \overline{\sigma_3}) \wedge (\overline{\sigma_1} \vee \sigma_2 \vee \sigma_3) \wedge \\ & (\overline{\sigma_1} \vee \overline{\sigma_2} \vee \overline{\sigma_3}) \wedge (\overline{\sigma_1} \vee \overline{\sigma_2} \vee \sigma_3) \wedge \\ & (\sigma_1 \vee \sigma_2 \vee \overline{\sigma_3}) \wedge (\sigma_1 \vee \sigma_2 \vee \sigma_3) \wedge \\ & (\sigma_1 \vee \overline{\sigma_2} \vee \overline{\sigma_3}) \wedge (\sigma_1 \vee \overline{\sigma_2} \vee \sigma_3) \end{aligned} \tag{B.1}$$

Supondo que as variáveis σ_1, σ_2 e σ_3 sejam respectivamente 1, 2 e 3, a expressão CNF γ mostrada na equação B.1[1] seria descrita como o arquivo mostrado na listagem B.1.

[1] Apenas a título de curiosidade, a fórmula γ descrita na referida equação é uma tautologia.

Listagem B.1. Exemplo de arquivo DIMACS.

```
c Arquivo CNF DIMACS. Esta linha é um comentário
p cnf 3 8
-1 2 -3 0
-1 2 3 0
-1 -2 -3 0
-1 -2 3 0
1 2 -3 0
1 2 3 0
1 -2 -3 0
1 -2 3 0
```