RESOLVEDOR MODULAR DE SATISFABILIDADE APLICADO NA VERIFICAÇÃO DE CIRCUITOS COMBINACIONAIS

BERNARDO CUNHA VIEIRA

RESOLVEDOR MODULAR DE SATISFABILIDADE APLICADO NA VERIFICAÇÃO DE CIRCUITOS COMBINACIONAIS

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
Co-Orientador: Fabrício Vivas Andrade

Belo Horizonte, MG

Fevereiro de 2010

Vieira, Bernardo Cunha
V658r Resolvedor modular de satisfabilidade aplicado na verificação de circuitos combinacionais / Bernardo Cunha Vieira. — Belo Horizonte, MG, 2010
xx, 86 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais
Orientador: Antônio Otávio Fernandes
Co-Orientador: Fabrício Vivas Andrade

1. Cálculo proposicional - Teses. 2. Circuitos integrados - Verificação - Teses. I. Orientador II. Co-Orientador III. Título

CDU 519.6*12.3(043)

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

FOLHA DE APROVAÇÃO

Resolvedor modular de satisfabilidade aplicado na verificação de circuitos combinacionais

BERNARDO CUNHA VIEIRA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

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

PROF. FABRÍCIO VIVAS ANDRADE - Co-orientador
Centro Federal de Educação Tecnológica de Minas Gerais - CEFET

PROF. CLAUDIONOR JOSÉ NUNES COELHO JÚNIOR
Departamento de Ciência da Computação - UFMG

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

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

Belo Horizonte, 03 de março de 2010.

Este trabalho é dedicado ao Nélio, meu pai, e a Maria Gorete, minha mãe.

Agradecimentos

Agradeço, em primeiro lugar, ao meu pai, Nélio, minha mãe, Maria Gorete, e meu irmão, Henrique, pelo incentivo a ser sempre uma pessoa melhor. Ao meu tio Newton José Vieira, pela revisão de alguns dos meus textos, mesmo com tantos erros, e pela contribuição infinita durante o meu mestrado. Ao Antônio Otávio, pelo incentivo de entrar, continuar no mestrado, pelo bom humor e pelas revisões. Ao Fabrício, pelas diversas horas gastas em conversas, nas revisões dos textos e por várias idéias e contribuições. A Raissa, por compreender a falta de horas durante esses dois anos. Aos membros da banca, pela disponibilidade e pelas revisões. A paciência e ao companheirismo de todos aqueles que conviveram comigo durante o mestrado, amigos e familiares. E a Deus, a verdadeira fonte de sabedoria.

Resumo

Os resolvedores SAT atuais, como Chaff[Malik et al., 2001], zChaff[Zhang et al., 2001], BerkMin[Goldberg & Novikov, 2007], e Minisat[Eén & Sörensson, 2003] geralmente compartilham das mesmas heurísticas principais, como por exemplo: aprendizado de cláusulas de conflito, *backtracking* não cronológico, e a estrutura dos dois literais vigiados. Por outro lado, eles se diferenciam na remoção de cláusulas de conflito, bem como na heurística de decisão do próximo literal. Esta dissertação apresenta uma nova abordagem para a construção de resolvedores SAT. Ela é baseada em fórmulas na forma normal conjuntiva, e implementa diversas heurísticas, como as propostas por Goldberg e Novikov em BerkMin [Goldberg & Novikov, 2007], e em Equivalência de Circuitos Dissimilares[Goldberg & Novikov, 2003], e Niklas Eén and Niklas Sörensson no Minisat. O Minisat, que foi o ponto de partida para a abordagem proposta, foi reimplementado para prover um *framework* no qual novas heurísticas podem ser testadas pela simples descrição em arquivos XML, realmente facilitando e tornando mais rápida a geração de novos e diferentes resolvedores SAT. Para demonstrar a efetividade da abordagem, esta dissertação também propõe cinco instâncias do resolvedor SAT modular para um importante e complexo problema de SAT: o problema da Equivalência de Circuitos Combinacionais. A primeira instância é um resolvedor que utiliza as heurísticas do BerkMin e do artigo Circuitos Dissimilares, menos a de remoção de cláusulas aprendidas, que foi adaptada do Minisat; a segunda instância é uma modificação da primeira que chaveia entre as heurísticas de decisão do BerkMin e do Minisat em tempo de execução; a terceira instância utiliza as heurísticas do BerkMin e do Circuitos Dissimilares menos a de decisão e remoção das cláusulas aprendidas que são adaptadas do Minisat; a quarta instância utiliza todas as heurísticas do BerkMin e do Circuitos Dissimilares; e a última é uma modificação da primeira que chaveia entre as heurísticas de remoção de cláusulas aprendidas em tempo de execução. Os experimentos mostram que a primeira instância gera um resolvedor que é mais rápido que os resolvedores estado-da-arte BerkMin e Minisat para diversas instâncias do problema SAT escolhido.

Lista de Figuras

Lista de Tabelas

Tabela de Siglas

SAT	Satisfabilidade,	p. 1
CEC	Combinational Equivalence Checking,	p. 1
ROBDDs	Reduced Ordered Binary Decision Diagram,	p. 1
FNC	Forma Normal Conjuntiva,	p. 1
CI	Circuito Integrado,	p. 3
EDA	Eletronic Design Automation,	p. 10
RTL	Register Transfer Level,	p. 12
DPLL	Algoritmo Davis-Putnam-Logemann-Loveland. Algoritmo de satisfabilidade mais estendido nos resolvedores atuais,	p. 16
DLL	Algoritmo Davis-Logemann-Loveland, mesmo que DPLL,	p. 16
BCP	Boolean Constraint Propagation,	p. 17
UIP	Unique Implication Point,	p. 26
MOM	*Maximum Occurrences on clauses of Minimum sizes*,	p. 27
VSIDS	Variable State Independent Decaying Sum,	p. 27
CS	Common Specification - Generalização de Circuitos com Similaridades Estruturais,	p. 39

Sumário

Capítulo 1

Introdução

O objetivo deste trabalho é propor uma nova abordagem modular, na construção de resolvedores de satisfabilidade (SAT). Essa abordagem será aplicada em um importante e complexo problema SAT: Verificação de Equivalência de Circuitos Combinacionais, ou CEC (do inglês, *Combinational Equivalence Checking*)[1]. A verificação de equivalência consiste em determinar se dois circuitos representam a mesma função booleana. A modelagem desse problema pode ser feita basicamente de duas formas: por grafos canônicos (ROBDDs[2]) ou por satisfabilidade de lógica proposicional. Em grafos, a equivalência corresponde ao isomorfismo dos grafos de cada circuito. O tamanho do grafo é determinado pela ordem de escolha da expansão das variáveis[3]. No caso de lógica proposicional, um circuito *miter*[4] é criado e posteriormente codificado na Forma Normal Conjuntiva (FNC). A fórmula na FNC é dada como entrada aos resolvedores de satisfabilidade, buscando-se pela insatisfabilidade, que prova que os dois circuitos originais são equivalentes. A Seção 1.1 explica que os multiplicadores não são eficientemente resolvidos por grafos.

1.1 Objetivo

Usando essa nova abordagem, foram implementadas as heurísticas do resolvedor Berk-Min [Goldberg & Novikov, 2002] e do artigo *Equivalence Checking of Dissimilar Circuits*[Goldberg & Novikov, 2003]. Como base utiliza-se o resolvedor Minisat, em

[1] Apesar da tradução correta de *Combinational* ser Combinatório, o termo Combinacional é muito utilizado na prática.

[2] *Reduced Ordered Binary Decision Diagram*

[3] Mais sobre BDDs em [Molitor & Mohnke, 2004] e [Clarke et al., 1999].

[4] *Miter* é um circuito em que a cada par de saídas primárias correspondentes é aplicada a função *XOR*. Mais na Seção 2.3.

[Eén & Sörensson, 2003], por ser um resolvedor de código aberto, compacto e vencedor de vários prêmios em competições, entre eles o SAT-Race 2006[5] e o SAT 2005 Competition[6]. Segundo os teste feitos por [Andrade, 2008], o resolvedor BerkMin foi o que obteve o melhor desempenho para diversas instâncias do problema CEC, como multiplicadores e divisores, com as entradas obtidas do gerador de testes BenCGen, [Andrade et al., 2008b]. Segundo [Molitor & Mohnke, 2004] e [Clarke et al., 1999], os multiplicadores não tem heurísticas de escolhas de literais que gerem ROBDDs menores que exponenciais. Desta forma, a abordagem mais prática para resolver instâncias com multiplicadores é a satisfabilidade.

Na questão modular, implementam-se as heurísticas do BerkMin e organizam-se as do Minisat em módulos intercambiáveis. A maior parte dos resolvedores não analisa e nem permite a combinação das diversas alterações propostas com o que já existe de outros resolvedores. Com essa nova abordagem, pretende-se criar um construtor de resolvedores. Os usuários serão capazes de escolher entre um conjunto de heurísticas disponíveis (inicialmente do BerkMin ou do Minisat) e criar, assim, um novo resolvedor a partir de qualquer combinação dos módulos. O conceito de heurística é definido na Definição 1.1.1.

Definição 1.1.1 *Heurísticas são métodos, baseados na experiência ou observações, que visam melhorar o desempenho de um programa.*

A Figura 1.1 representa melhor a modularização e o objetivo pretendido. Cada quadro representa um módulo do resolvedor. Ao se implementar as heurísticas do BerkMin de forma modular sobre o Minisat, tem-se dois resolvedores completos: o BerkMin (representado em verde) e o Minisat (representado em azul). Com a nova abordagem, pode-se criar novos resolvedores combinando-se alguns dos módulos. O Resolvedor X da figura não existia, e inclui todas as heurísticas do BerkMin, menos a *ReduceDB* que é retirada do Minisat, com as devidas modificações. Mais explicações sobre cada quadro, isto é, sobre as heurísticas, podem ser encontradas na Seção 3.3.

1.2 Motivação

O desenvolvimento de circuitos está intimamente ligado ao conceito *time-to-market*, que basicamente indica que um circuito lançado após certo prazo perde mercado. Até mesmo produtos mais novos e avançados podem ser lançados antes do circuito atrasado.

[5] `http://fmv.jku.at/sat-race-2006/`
[6] `http://www.satcompetition.org/2005/`

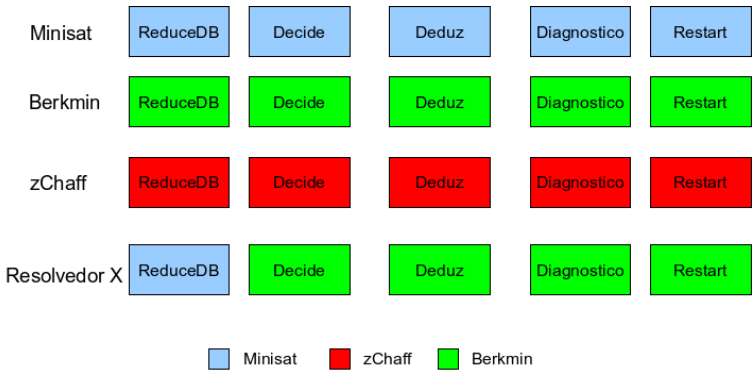


Figura 1.1. Resolvedores de satisfabilidade e seus módulos

Isso gera grandes prejuízos financeiros. Logo, há grandes restrições quanto ao tempo de desenvolvimento dos CIs, circuitos integrados.

Outro fator que pode causar prejuízo são falhas nos circuitos; para evitá-las, eles devem ser validados. É consensual que a maior parte do tempo de desenvolvimento do circuito é gasto na validação, [Ozguner et al., 2001]. Além disso, a cada ano, seguindo a Lei de Moore[Moore, 2006], os circuitos possuem mais transistores, e por consequência mais lógica, o que contribui para o aumento do tempo de verificação. Acrescenta-se que nos últimos tempos, novos CIs são utilizados nos sistemas embutidos, como celulares. Portanto, mais circuitos com maior número de portas lógicas precisam ser verificados em menos tempo.

Os principais métodos de validação são simulação, testes, verificação dedutiva e verificação automática. A satisfabilidade se aplica a vários contextos na validação de circuitos integrados, utilizável tanto em testes, quanto em simulação e verificação automática. Em testes há, por exemplo, a Geração Automática de Padrões de Testes, como feito por [Larrabee, 1992]. Nesse artigo é introduzido algum erro do tipo *stuck-at-1* ou *stuck-at-0*[7]. As atribuições feitas pelo resolvedor de satisfabilidade são as entradas que devem ser aplicadas ao circuito para verificação deste tipo de erro. Essa abordagem limita o uso desse método de geração de testes, já que são específicos para a área em que ocorre o *stuck-at*. No campo de verificação, há a verificação lógica, como por exemplo testar se um circuito otimizado é equivalente a um não otimizado. Em simulação, existe a possibilidade de análise de tempo através de satisfabilidade[8]. Na Seção 2.6 é explicitado que o resolvedor proposto é utilizado na fase de verificação lógica.

[7]Os erros *stuck-at-n* indicam que uma parte do circuito fica em curto circuito (n=0) ou ligada a VCC (n=1)

[8]Mais sobre a importância de SAT em circuitos integrados estão descritas em [Marques-Silva & Sakallah, 2000].

1.3 Contribuições

Este trabalho, ao contrário do BerkMin, será disponibilizado em código aberto, já que um dos objetivos é facilitar a implementações de novas heurísticas. Além disso, a abordagem proposta é capaz de gerar novos resolvedores, já que permite a escolha de quais módulos serão utilizados. A partir do resolvedor desenvolvido três hipóteses são testadas, visando a diminuição do tempo de execução:

1. Mudar o BerkMin e o artigo Circuitos Dissimilares para usar o método de redução da base de cláusulas do Minisat;

2. Usar o BerkMin e o artigo Circuitos Dissimilares como base, chavear, em tempo de execução, as heurísticas de decisão entre a do Minisat e a do BerkMin e utilizar o método de redução da base de cláusulas do Minisat;

3. Usar o BerkMin e o artigo Circuitos Dissimilares como base e chavear, em tempo de execução, a redução da base de cláusulas entre a do Minisat e a do BerkMin.

Outras duas instâncias do resolvedor proposto, sem hipóteses geradoras, são também apresentadas. Uma possui todas as heurísticas do BerkMin e do artigo Circuitos Dissimilares, enquanto a outra usa as heurísticas do Berkmin e do artigo Circuitos Dissimilares menos as heurísticas de escolha do próximo literal e a de redução da base de cláusulas, que são adaptadas do Minisat.

Acredita-se que implementar e testar as heurísticas apresentadas pelo BerkMin, ajuda também a entender o problema, já que foi ele que obteve os melhores resultados experimentais para os circuitos criados e testados por [Andrade et al., 2008b].

1.4 Organização

No Capítulo 2, foram apresentados os principais conceitos, como satisfabilidade e equivalência de circuitos para a posterior apresentação da revisão bibliográfica, feita no Capítulo 3. O Resolvedor Modular de Satisfabilidade Aplicado na Verificação de Circuitos Combinacionais é mostrado no Capítulo 4, com os resultados no Capítulo 5 e a conclusão e trabalhos futuros no Capítulo 6.

Capítulo 2

Conceitos Básicos

A Seção 2.1 define o problema de satisfabilidade (SAT). A Seção 2.2 descreve as Formas Normais Conjuntiva (FNC) e Disjuntiva (FND). A FNC é geralmente utilizada como entrada dos resolvedores de satisfabilidade. A Seção 2.3 define o que é o problema de Equivalência de Circuitos Combinacionais (CEC) e como ele é modelado. A Seção 2.4 disserta sobre os *benchmarks* e qual foi escolhido. Já a Seção 2.5 explica o principal formato utilizado pelos *benchmarks*. A Seção 2.6 mostra um pouco sobre desenvolvimento de circuitos integrados e em que parte desse desenvolvimento este trabalho é utilizado.

2.1 Satisfabilidade

Satisfabilidade (Definição 2.1.2) é a busca de uma atribuição de variáveis proposicionais que torna uma fórmula proposicional verdadeira, ou determinar que tal atribuição não existe[1].

Uma pessoa, quando modela, cria um modelo conceitual. Esse modelo é um conjunto de abstrações da realidade[2]. Elas contém tudo aquilo que é estritamente essencial para descrever a realidade, e não contém detalhes supérfluos que possam prejudicar o entendimento do modelo (e na computação, também a eficiência computacional). Há várias entidades matemáticas para a modelagem, como por exemplo conjuntos, grafos, equações diferenciais. Na computação, após obtido um modelo conceitual, o progra-

[1]Durante o texto, letras gregas ou a letra maiúscula F, representam uma fórmula proposicional, e a função $v(\alpha)$ representa o valor da fórmula proposicional α.

[2]No contexto da computação, essas abstrações representam a visão da realidade, ou da pessoa que modela ou dela e de seus clientes. A realidade, principalmente na computação, não necessariamente é constituída de objetos reais, podendo conter entidades abstratas, representando qualquer coisa que povoe a realidade da aplicação.

mador deve codificar o modelo em uma linguagem formal, passível de ser tratada por um computador.

Essa representação do conhecimento é explicada na Figura 2.1. Um objeto do mundo real é abstraído por um conceito no nível conceitual, que é denotado por um símbolo no nível formal. Detalhes são omitidos na abstração. O processo de abstração é denominado modelagem e o processo de denotação é denominado formalização. Os processos inversos são chamados de interpretação e indicam a semântica.

Resumindo e citando o exemplo do lógica proposicional, a representação lógica do conhecimento, é composta de dois níveis além da realidade. O nível conceitual fornece a semântica (verdadeiro, falso, e o que representa cada conectivo lógico), que é dependente da realidade. O nível formal fornece a representação em linguagem formal dos conceitos (proposições e conectivos) identificados no nível conceitual.

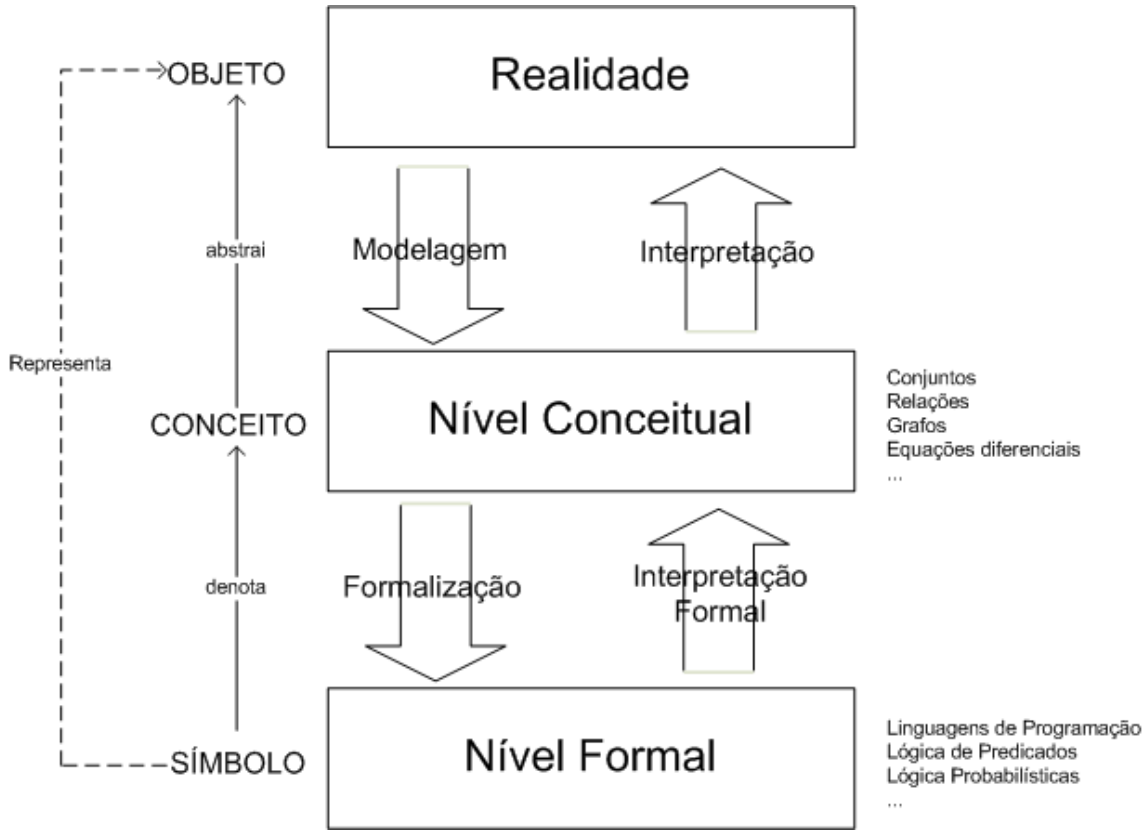


Figura 2.1. Os processos de Modelagem e Formalização [Vieira, 2008].

No nível conceitual da lógica proposicional, uma proposição, ou fórmula proposicional (α) é um enunciado, que pode ser verdadeiro ou falso ($v(\alpha) = V$, ou $v(\alpha) = F$, respectivamente). As proposições podem ser simples ou compostas. As proposições simples não são divisíveis, já as compostas são constituídas de outras proposições,

denominadas sub-fórmulas, ligadas por algum conectivo lógico (basicamente negação, conjunção, disjunção). Essas sub-fórmulas são também proposições simples ou compostas. O valor de uma fórmula proposicional composta é determinado pela semântica de cada conectivo ($f_b(con)\alpha, \beta$[3]). Essa semântica é geralmente representada através da construção de uma tabela da verdade, como a Tabela 2.1. A tabela da verdade para uma proposição composta é construída recursivamente através das tabelas da verdade de cada sub-fórmula e da semântica de seu conectivo lógico.

Conforme dito, cada representação do conhecimento no nível conceitual tem sua correspondente no nível formal, e vice-versa. O nível formal é o nível utilizado no processamento computacional. Nesse nível, as variáveis proposicionais são símbolos (a, b, p, q) e representam as proposições simples. Cada conectivo lógico tem sua respectiva representação formal ($\neg, \wedge, \vee$, representando negação, conjunção, disjunção). A propagação de valores representada pela tabela da verdade, é implementada no nível formal, através de regras de simplificação, que pode ser encontrado em [Vieira, 2008]. A Definição 2.1.1 é a definição, no nível formal de fórmulas proposicionais. Essas fórmulas proposicionais, do nível formal, geralmente escritas na FNC, são as entradas para os resolvedores de satisfabilidade.

$v(\alpha)$	$v(\beta)$	$f_b(\neg)\alpha$	$f_b(\neg)\beta$	$f_b(\wedge)\{\alpha, \beta)\}$	$f_b(\vee)\{\alpha, \beta\})$	$f_b(\rightarrow)\{\alpha, \beta\})$
F	F	V	V	F	F	V
F	V	V	F	F	V	V
V	F	F	V	F	V	F
V	V	F	F	V	V	V

Tabela 2.1. Tabela da Verdade para alguns conectivos lógicos.

Definição 2.1.1 *São fórmulas proposicionais, [Vieira, 2008]:*

- *Cada variável proposicional é uma fórmula (simples).*
- $\top$ *e* $\bot$ *são fórmulas (representam verdadeiro e falso no nível conceitual).*
- *Se* α *é uma fórmula então* $\neg\alpha$ *é uma fórmula.*
- *Se* α *e* β *são fórmulas então,* $\alpha \wedge \beta$, $\alpha \vee \beta$, $\alpha \rightarrow \beta$ *e* $\alpha \leftrightarrow \beta$ *são fórmulas.*
- *Só são fórmulas as fórmulas descritas acima.*

[3]Onde *con* é um conectivo lógico, α e β são fórmulas proposicionais e f_b é uma função $(\alpha, \beta) \rightarrow \{V, F\}$.

Percebe-se pela Definição 2.1.2 que a satisfabilidade envolve a busca por uma interpretação (Definição 2.1.3) das variáveis, isto é, por um modelo. Um algoritmo de força bruta pode ser facilmente desenvolvido para achar modelos. Basta testar todas as combinações de atribuições de variáveis, fazer a simplificação, e verificar quais delas satisfazem à fórmula. Nesse algoritmo, o espaço de busca é muito grande (2^n, onde n é o número de variáveis proposicionais da fórmula).

Definição 2.1.2 *[Vieira, 2008] Uma fórmula α é satisfazível se, e somente se, α tem um modelo. Se α não é satisfazível (não tem modelo), então é insatisfazível (ou uma contradição).*

Definição 2.1.3 *Modificado de [Vieira, 2008].*
Uma interpretação i é um modelo para uma fórmula α se, e somente se, o significado da fórmula α seguindo a interpretação i é verdadeiro. Os modelos são as linhas da tabela da verdade que tornam a fórmula verdadeira, ou que, no nível formal, após as simplificações e escolhas de variáveis retorne o literal $\top$.

2.2 Formas Normais

A maior parte dos resolvedores de SAT recebem como entradas fórmulas proposicionais codificadas na FNC (Definição 2.2.1). O elemento básico das formas normais é o literal. O literal é ou uma variável proposicional[4], ou a negação de uma variável proposicional[5].

Definição 2.2.1 *[Vieira, 2008] Uma fórmula α está na forma normal conjuntiva (FNC), se é $\top$, $\bot$, ou da forma $\phi_1 \wedge \phi_2 \wedge \cdots \wedge \phi_n$, em que $n \geq 1$, e cada ϕ é uma cláusula (Definição 2.2.2) não vazia.*

Definição 2.2.2 *[Vieira, 2008] Uma cláusula é uma disjunção de literais $l_1 \vee l_2 \vee \cdots \vee l_n$, n$\geq$ 0. No caso em que n = 0, tem-se a denominada cláusula vazia, que é denotada por $\bot$ (contradição).*

Definição 2.2.3 *[Vieira, 2008] Uma fórmula α está na forma normal disjuntiva (FND), se é $\top$, $\bot$, ou da forma $\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n$, em que $n \geq 1$, e cada ϕ é uma cláusula dual (Definição 2.2.4) não vazia.*

[4]Nesse caso o literal é positivo (p)
[5]Nesse caso o literal é negativo ($\overline{p}$).

Definição 2.2.4 *[Vieira, 2008] Uma cláusula dual é uma disjunção de literais* $l_1 \wedge l_2 \wedge \cdots \wedge l_n$, $n \geq 0$. *No caso em que* $n = 0$, *tem-se a denominada cláusula dual vazia, que é denotada por* $\top$ *(tautologia, Definição 2.2.5).*

Definição 2.2.5 *[Vieira, 2008] Uma fórmula* α *é uma tautologia se, e somente se, todas as interpretações tornam* α *verdadeira.*

A fórmula na forma normal disjuntiva (FND - Definição 2.2.3) é facilmente satisfazível (basta satisfazer uma cláusula dual). Porém a conversão de um circuito para FND, não é trivial, como é para FNC. Há métodos de conversão de uma fórmula na forma normal conjuntiva para uma fórmula normal disjuntiva e vice-versa. Estes métodos utilizam a distribuição do $\wedge$ com relação ao $\vee$ (FNC $\rightarrow$ FND) ou do $\vee$ com relação a $\wedge$ (FND $\rightarrow$ FNC), e resultam em complexidade exponencial no pior caso, de memória e/ou de tempo. Logo, a transformação pura e simples para FND não é utilizada.

2.3 Equivalência de Circuitos

A verificação de equivalência de dois circuitos combinacionais consiste em provar a equivalência funcional dos circuitos, isto é, dada qualquer combinação de entrada, os circuitos devem gerar a mesma saída. Segundo [Molitor & Mohnke, 2004], o problema pode ser formalizado como:

> *Dadas duas representações* d_f *e* d_g *de duas funções booleanas f, g:* $\{0,1\}^n \rightarrow \{0,1\}^m$, *decida se as duas funções f e g são iguais, isto é, se* $f(\alpha) = g(\alpha)$ $\forall \alpha \in \{0,1\}^n$.

Avalia-se, a seguir, as maneiras de se testar se um circuito é equivalente a outro através de satisfabilidade. Pode-se, por exemplo, utilizar o operador bicondicional entre as fórmulas FNC de cada circuito. Basta provar que o circuito resultante é uma tautologia, que a equivalência é garantida. Porém, geralmente para o problema CEC uma outra abordagem é mais utilizada. Utiliza-se um circuito denominado de *miter*, que é um circuito em que a cada par de saídas primárias correspondente de cada circuito é aplicada a função *XOR* e à saída de cada porta XOR é aplicada a função *OR*. Um circuito *miter* é mostrado na Figura 2.2. A transformação do circuito *miter* em uma FNC para entrada de um resolvedor de satisfabilidade é bastante simples. Basta utilizar a Tabela 2.2. Se os circuitos não são equivalentes, a saída S da Figura é eventualmente 1, então a FNC é satisfazível. Caso seja equivalente, a saída S da Figura é sempre 0, e a fórmula é insatisfazível. Essa transformação retira os aspectos

estruturais do circuito, como a distância até as entradas e a propagação paralela de valores segundo a função da porta lógica (que passa a ser feita seqüencialmente pelo BCP. Para o BCP, veja Seção 3.1.).

Tipo Da Porta	**Função**	φ_x
AND	$x = AND(w_1, \cdots, w_j)$	$\left[\prod_{i=1}^{j}(w_i + \neg x)\right] . \left(\sum_{i=1}^{j} \neg w_i + x\right)$
NAND	$x = NAND(w_1, \cdots, w_j)$	$\left[\prod_{i=1}^{j}(w_i + x)\right] . \left(\sum_{i=1}^{j} \neg w_i + \neg x\right)$
OR	$x = OR(w_1, \cdots, w_j)$	$\left[\prod_{i=1}^{j}(\neg w_i + x)\right] . \left(\sum_{i=1}^{j} w_i + \neg x\right)$
NOR	$x = NOR(w_1, \cdots, w_j)$	$\left[\prod_{i=1}^{j}(\neg w_i + \neg x)\right] . \left(\sum_{i=1}^{j} w_i + x\right)$
NOT	$x = NOT(w_1)$	$(x + w_1) . (\neg x + \neg w_1)$
BUFFER	$x = BUFFER(w_1)$	$(\neg x + w_1) . (x + \neg w_1)$

Tabela 2.2. Tabela de conversão circuito → FNC [Marques-Silva & Sakallah, 2000]

2.4 Benchmarks

Um *benchmark* em EDA é um conjunto de circuitos codificados em um formato comum, utilizado para avaliar algoritmos e ferramentas, sobre um importante domínio do problema [Andrade et al., 2008b]. Geralmente os *benchmarks* possuem uma descrição para cada instância de qual domínio do problema ela se refere[6]. Há basicamente dois grupos de *benchmarks* para EDA, de acordo com o desenho que eles possuem. O primeiro são os *benchmarks* industriais e o segundo os *benchmarks* sintéticos. Para os *benchmarks* industriais de CEC, utilizando SAT, evidencia-se o uso dos *benchmarks* ISCAS85 e ISCAS89. [Andrade et al., 2008b] aponta três problemas para a utilização dos *benchmarks* industriais, como o ISCAS85. O primeiro e mais importante problema é que as companhias de circuitos integrados não proporcionam as descrições atualizadas dos circuitos devido a propriedade intelectual; são como segredos industriais. Dessa forma, os *benchmarks* industriais ficam rapidamente desatualizados. O segundo é que

[6]Um conjunto grande de *benchmarks* de variados de diversos domínios não relacionados a EDA pode ser encontrado em `http://people.cs.ubc.ca/~hoos/SATLIB/benchm.html`. Geralmente para o domínio CEC, quando são utilizado resolvedores de SAT, o *miter* do circuito é codificado no formato DIMACS.

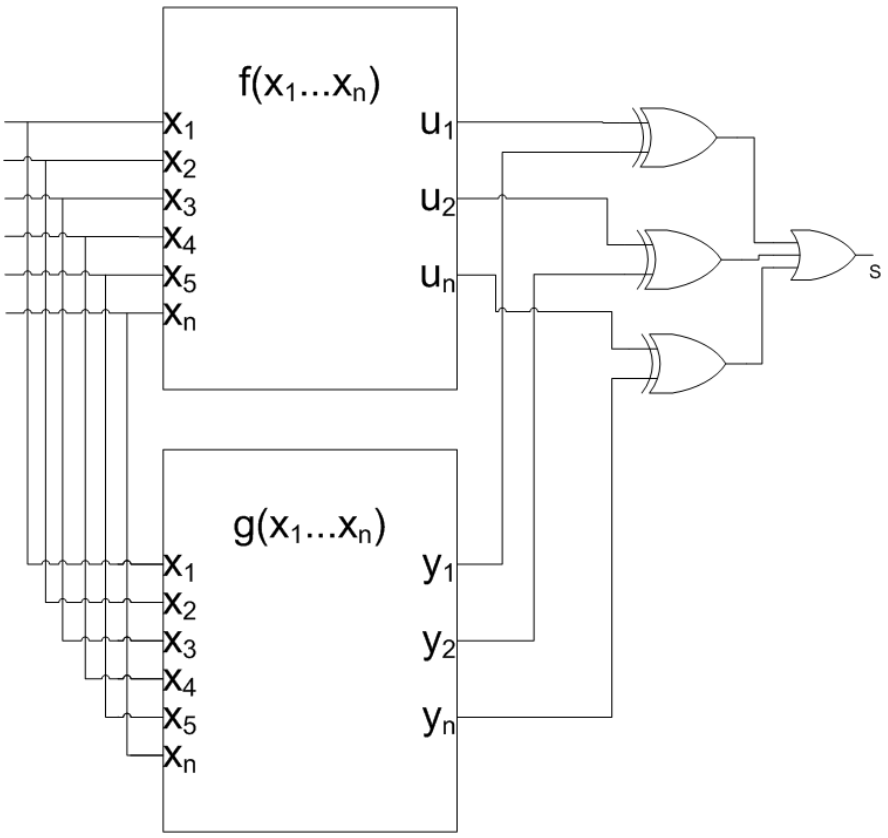


Figura 2.2. Circuito Miter.

somente uma quantidade bem pequena dos circuitos são tornados públicos, também devido a propriedade intelectual. A terceira razão é que geralmente os circuitos tem tamanho limitado e geralmente de poucos bits, o que atrapalha na avaliação pois um algoritmo pode ser bom para 16 bits e não ser para 32 bits.

Desta forma, escolheu-se o *benchmark* oferecido por [Andrade et al., 2008b], chamado BenCGen. Este *benchmark* é um gerador parametrizável de circuitos, entre eles multiplicadores, divisores e somadores. Com esta ferramenta foram gerados circuitos de tamanho 2 a 32. Para cada circuito transformado em *netlists* (chamado C'), foi gerado um circuito C'' otimizado pela ferramenta ABC[Mishchenko, 2009]. O circuito *miter* foi gerado a partir de C' e C''. O miter é transformado no formato DIMACS utilizado-se a Tabela 2.2[7].

[7]Para *netlist* veja a seção 2.6 e para a transformação de portas lógicas em FNC veja seção 2.3.

2.5 DIMACS

O DIMACS é um padrão de descrição de uma fórmula lógica na FNC, e é utilizado pela maioria dos resolvedores de SAT e pelo menos por todos resolvedores que serão utilizados no Capítulo 5, inclusive o implementado. O arquivo pode começar com comentários. Neste caso cada linha possui no início a letra c minúscula. Após os comentários, vem uma linha começada com a expressão "p cnf" seguido do número de variáveis e o número de cláusulas, separados por espaço. Cada linha posterior representa uma cláusula. Os literais positivos são representados por número positivos, os literais negativos pelos números negativos e o zero representa o fim da cláusula. A variável é representada pelo valor absoluto do literal. O número zero representa o fim da cláusula.

Exemplo 2.1 Exemplo de arquivo no formato DIMACS

```
c Um exemplo.
p cnf 6 2
-1 2 -3 0
 1 4  5 0
```

2.6 Fabricação de Circuitos

Para melhor entender o processo geral de fabricação de circuitos integrados, adapta-se a seguir os passos típicos de implementação de um circuito usando FPGAs. FPGAs são dispositivos lógicos programáveis. O ciclo básico é descrito no fluxograma da Figura 2.3.[8]

A fabricação de circuitos começa em um projeto de alto nível, com as principais funcionalidades do sistema. Implementa-se, então, a especificação em uma linguagem de alto nível como Verilog ou VHDL. Testes são feitos em simuladores para validação inicial dos circuitos. Na simulação, além do aspecto funcional, podem ser verificados alguns aspectos de comportamento, como condições de corrida e temporização (quadro *Análise de Tempo e Simulação*). Durante toda a fase de desenvolvimento, os circuitos são retestados após qualquer alteração. Os testes, neste caso, são chamados de testes de regressão.

Geralmente a modelagem feita em RTL (*Register Transfer Level*, como o código Verilog) passa por uma série de transformações antes de ser implementada no circuito integrado. No primeiro processo, chamado de síntese, são geradas as descrições dos

[8]O documento original é encontrado na documentação do software Quartus II 9.0 [Altera, 2009].

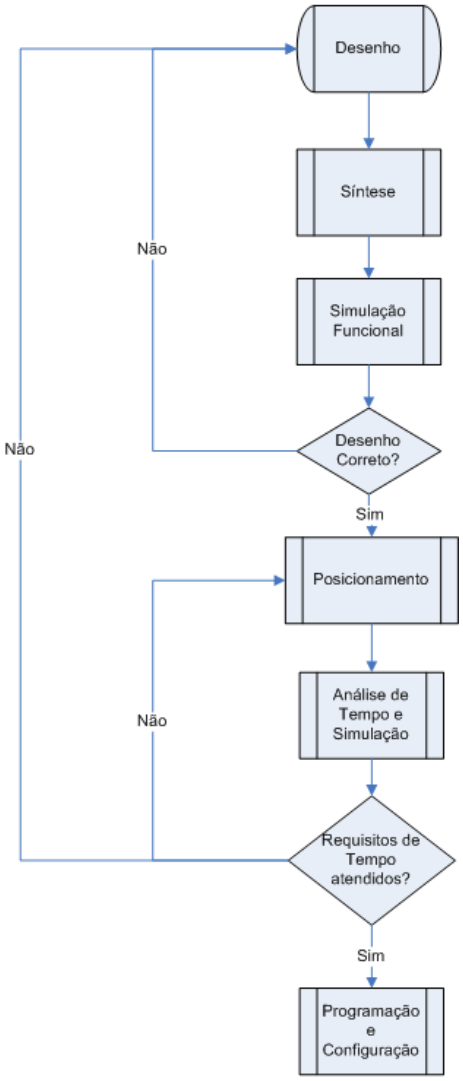


Figura 2.3. Fluxograma de um projeto de circuito integrado[Altera, 2009].

circuitos envolvidos em portas lógicas. Teoricamente, cada módulo poderia ter sido inicialmente desenvolvido apenas em portas lógicas, mas o desenvolvimento fica muito mais lento e tem mais chance de conter erros. Os modelos apenas com portas lógicas são chamados de listas de portas lógicas (*gate-level netlists*). O circuito sintetizado pode então sofrer otimizações e mapeamento tecnológico, gerando um novo circuito. Finalmente o circuito sintetizado e otimizado é transformado em um circuito físico. Este trabalho garante que a transformação, feita entre a fase de síntese inicial e a fase de otimização, gere um circuito equivalente ao anterior à transformação.

Capítulo 3

Revisão Bibliográfica

Este capítulo é composto principalmente de dissertações, teses e artigos relacionados a resolvedores de Satisfabilidade (SAT), alguns com contribuições para grandes classes de problemas, como o Chaff, e outros relacionados às áreas mais específicas, como o *Dissimilar Circuits* é para CEC (*Combinational Equivalence Checking*). Conforme se observa, os resolvedores vão acumulando as contribuições de resolvedores anteriores e melhorando o tempo de execução com novas heurísticas e estruturas de dados. Portanto, para se entender um resolvedor, deve-se entender os resolvedores anteriores. Muitas vezes, não se pode dizer quais são os pontos negativos de um resolvedor, pois o método ou estrutura de dados nem existia ainda, mas apenas o que foi melhorado. Isso é evidente observando-se resolvedores para instâncias de problemas específicos, já que eles vencem os de uso geral (nessas instâncias pelo menos).

Um fator interessante em utilizar SAT para CEC é que novas heurísticas, novos arranjos de estruturas de dados e até mesmo novos resolvedores utilizados para outra classe de problemas podem ser testados em CEC. Dessa forma, modelando o problema CEC na FNC, pode-se aplicá-lo como entrada aos diversos resolvedores e ver quais deles tem o melhor tempo de resposta.

Uma revisão das principais heurísticas e módulos implementados nos resolvedores SAT estado-da-arte baseados em DPLL são mostrados por [Kautz & Selman, 2007]. Para demonstrar a herança entre os resolvedores recomenda-se uma leitura cuidadosa desses resolvedores nesta ordem: DPLL[Davis et al., 1962], GRASP[Marques-Silva & Sakallah, 1999], SATO[Zhang, 1997], Chaff[Malik et al., 2001], zChaff[Zhang et al., 2001], BerkMin[Goldberg & Novikov, 2007] e Minisat[Eén & Sörensson, 2003][1] Por exemplo,

[1]O DPLL é a base, com o método BCP. O aprendizado de cláusulas de conflito e *backtracking* não cronológico foram usados e explicados no GRASP. O *backtracking* não cronológico é introduzido por

a estrutura "Two-Watched Literals", ou "Watched Literals", melhorou bastante o tempo de execução em comparação às outras estruturas de dados. A estrutura de literais vigiados foi proposta por SATO, e modificado pelo Chaff com a eliminação da restrição cabeça e cauda[2]. Outro exemplo, os diferentes métodos de aprendizagem de pontos de implicação (UIPs) e o esquema 1-UIP são apresentados no resolvedor zChaff. Por fim novas heurísticas para problemas gerais ou específicos são implementadas pelos resolvedores Chaff (geral), BerkMin (específico CEC) e Minisat (geral). Por exemplo: Chaff com VSIDS, BerkMin com a base de cláusulas aprendidas como uma pilha, e Minisat com a redução da base de dados com decaimento da atividade da cláusula. Todos esses conceitos serão apresentados e explicados durante este capítulo. As seções seguintes explicam as principais contribuições desses resolvedores.

3.1 DPLL

Praticamente todos os resolvedores de SAT atuais utilizam alguma variante sobre o método DPLL, descrito nesta seção. O trabalho é composto de dois artigos. O primeiro artigo, dos autores Davis e Putnam, foi sobre um método denominado DP [Davis & Putnam, 1960]. O artigo que descreve o DPLL foi escrito por Davis, Logemann e Loveland, o que leva muitos autores a se referirem ao método como DLL, ou DPLL, lembrando da contribuição de Putnam no primeiro artigo. No DP, os autores descrevem um algoritmo para o problema de lógica de predicados, mas o que realmente interessa neste artigo é o algoritmo para satisfabilidade de sequências de linhas livres de quantificadores (em que as fórmulas quantificadas já foram transformadas em fórmulas proposicionais). O método DP é mostrado no Algoritmo 3.1 em três regras principais. A prova para manutenção de inconsistência entre F e F' do Algoritmo 3.1 é apresentada no artigo e envolve resolução. Um sério problema é a possível explosão exponencial do tamanho da fórmula no número de variáveis, o que pode ocorrer na fase III. Após a eliminação da variável, a fórmula deve ser reescrita na FNC. Nesse passo, cada fórmula de A deve ser resolvida na variável p, com todas as cláusulas de B que contenham $\overline{p}$. A cada eliminação de variável pode-se gerar mn cláusulas, onde m é o tamanho de A e n é o tamanho de B. Isto fornece uma equação quadrática (n^2) a cada eliminação. A equação de recorrência, Equação 3.1, onde n é o número de variáveis eliminadas, apresenta a complexidade desta operação. Apesar de o pior caso nem sempre ocorrer, devido a redução da fórmula causada pelo BCP e pela eliminação de literais puros,

[Stallman & Sussman, 1977].

[2]Apesar do termo "Watched Literals" ser o mais utilizado, preferiu se referir à estrutura como "Two-Watched Literals" para evidenciar o uso de dois literais vigiados.

esta possibilidade representa um problema. Retirou-se o Exemplo 3.1, da regra III do artigo do DP para demonstrar a aplicação dessa regra.

Algoritmo 3.1 Algoritmo DP [Davis & Putnam, 1960]

I Eliminação de cláusulas com um literal (BCP):

- a Se uma fórmula F na FNC contém a cláusula de um literal p e também contém a cláusula de um literal $\neg p$ então F é contraditória.
- b Se (a) não se aplica então, se a cláusula p aparece em F, então modifique F retirando todas as cláusulas que contenham p; e removendo $\overline{p}$ de todas as outras cláusulas, obtendo F' que é inconsistente se e somente se F é inconsistente.
- c Se (a) não se aplica então, se a cláusula $\overline{p}$ aparece em F, então modifique F retirando todas as cláusulas que contenham $\overline{p}$; e removendo p de todas as outras cláusulas, obtendo F' que é inconsistente se e F é inconsistente.
- d Em (b) e (c) se F' é vazio, F é consistente.

II Afirmativa-Negativa (Literais Puros): se p ocorre apenas afirmativamente em F retire de F todas as cláusulas que contenham p. Se p ocorre apenas negativamente em F retire de F todas as cláusulas que contenham $\overline{p}$. A fórmula resultante é inconsistente se e somente se F é inconsistente.

III Eliminação de fórmulas atômicas (resolução de um literal). Seja a fórmula F escrita na forma $(A \vee p) \wedge (B \vee \overline{p}) \wedge R$, em que A, B, e R são livres de p. Isto pode ser feito agrupando as cláusulas que contenham p e fatorando as ocorrências de p, obtendo A. O mesmo pode ser feito para $\overline{p}$, obtendo B. As outras cláusulas são agrupadas fornecendo R. F é inconsistente se e somente se $(A \vee B) \wedge R$ é inconsistente.

Algoritmo 3.2 DPLL

Entrada: α - fórmula proposicional

```
1: α := bcp(α)
2: α := literal − puro(α)
3: se α tem cláusula vazia então
4:    retorne falso
5: fim se
6: p := escolha − literal(α)
7: se p == nulo então
8:    retorne verdadeiro
9: fim se
10: retorne DPLL(α ∧ p) || DPLL(α ∧ p̄)
```

O DPLL, uma nova versão para o DP, continua usando BCP, literais puros, porém usa uma regra *splitting* ao invés da regra III do DP. O DPLL, ou DLL

Equação 3.1 Equação de recorrência para a regra III do DP

Equação recursiva:

$$\begin{aligned} T(1) &= 2 \\ T(n) &= T(n-1)^2 \end{aligned}$$

Resolução:

$$\begin{aligned} T(2) &= T(1)^2 = 2^2 \\ T(3) &= T(2)^2 = T(1)^{(2*2)} \\ T(4) &= T(3)^2 = T(1)^{(2*2*2)} \\ &\vdots \\ T(n) &= T(1)^{(2^{(n-1)})} = 2^{(2^{(n-1)})} \\ O(T(n)) &= 2^{(2^n)} \end{aligned}$$

Exemplo 3.1 Exemplo da aplicação da regra III do DP [Davis & Putnam, 1960]

$$\begin{aligned} &(p \vee r) \wedge (p \vee \overline{s}) \wedge (\overline{p} \vee s) \wedge (\overline{p} \vee \overline{r}) \wedge (s \vee \overline{r}) \wedge (\overline{s} \vee r) \\ &((r \wedge \overline{s}) \vee p) \wedge ((\overline{r} \wedge s) \vee \overline{p}) \wedge (s \vee \overline{r}) \wedge (\overline{s} \vee r) \\ &((r \wedge \overline{s}) \vee (\overline{r} \wedge s)) \wedge (s \vee \overline{r}) \wedge (\overline{s} \vee r) \ (\textit{Regra III } p \textit{ eliminado}) \\ &(r \vee \overline{r}) \wedge (s \vee r) \wedge (\overline{s} \vee \overline{r}) \wedge (s \vee \overline{s}) \wedge (s \vee \overline{r}) \wedge (\overline{s} \vee r) \ (\textit{Regra III distribuição}) \\ &(s \vee r) \wedge (\overline{s} \vee \overline{r}) \wedge (s \vee \overline{r}) \wedge (\overline{s} \vee r) \\ &((s \wedge \overline{s}) \vee r) \wedge ((s \wedge \overline{s}) \vee r) \\ &(s \wedge \overline{s}) \ (\textit{Regra III } r \textit{ eliminado}) \end{aligned}$$

[Davis et al., 1962], diferencia-se do seu antecessor [Davis & Putnam, 1960] na troca de uma possível explosão exponencial no espaço por possível explosão exponencial no tempo, enquanto conserva um espaço linear na memória.

> Regra *splitting*, como formulado em [Davis et al., 1962]: *"Seja a fórmula $\mathcal{F}$ reescrita na forma $(A \vee p) \wedge (B \vee \overline{p}) \wedge R$, onde A, B e R não possuem p ou $\overline{p}$, isto é são livres de p. Então $\mathcal{F}$ é inconsistente se e somente se $A \wedge R$ e $B \wedge R$ são ambos inconsistentes. Justificativa: Se p=0, $\mathcal{F} = A \wedge R$, se p=1 $\mathcal{F} = B \wedge R$"*

A regra *splitting* é representada pela chamada recursiva encontrada na última

linha do algoritmo DPLL, representado no Algoritmo 3.2. O artigo prova que a inconsistência da fórmula é mantida.

A troca da explosão de espaço por tempo, levo o DPLL a servir como base para a maioria dos resolvedores de SAT baseados em FNC de uso geral atuais. A regra de literais-puros não é geralmente utilizada, principalmente ao alto custo de verificação (deve-se percorrer todas as cláusulas e verificar que o literal aparece apenas com um sinal). O Algoritmo 3.2 escolhe um literal livre p e testa os ramos dos assinalamentos da variável p. Se, e somente se, o assinalamento não resultar em satisfabilidade, então p é assinalado como $\overline{p}$, e o DPLL opera sobre o novo ramo. Se ambos são insatisfazíveis então um assinalamento anterior deve ser refeito. Se, e somente se, todos os assinalamentos forem insatisfazíveis, então a fórmula é insatisfazível.

3.1.1 Pilha de Assinalamentos

Devido à recursão de cauda, geralmente implementa-se o algoritmo com uma pilha de assinalamentos. Marcadores são colocados na pilha quando ocorre uma decisão. O evento de assinalamento de qualquer variável (isto é, a variável e o valor) é sempre empilhado. Quando ocorre um conflito, os assinalamentos são desempilhados até o nível desejado, representados pelos marcadores. No caso do DPLL, basta desfazer os assinalamentos até o primeiro marcador. Por outro lado, no caso de um *backtracking* não cronológico deve-se desfazer os assinalamentos e os marcadores até o nível desejado. O nível de decisão 0 representa os assinalamentos iniciais do problema devido ao BCP.

A Figura 3.1 mostra um caso de uso da pilha de assinalamentos. Os marcadores são representados por "-". Os assinalamentos de decisão, sublinhados na parte superior da figura, são empilhados após os marcadores. Os outros assinalamentos são feitos pelo BCP. O caso F representa um conflito, que resulta no desempilhamento até o marcador mais próximo.

3.1.2 Refutação por Resolução Generalizada

Conforme [Ryan, 2002] e [Gomes et al., 2008], os algoritmos atuais são semelhantes ao DPLL, mas tem uma característica diferente muito interessante. Eles podem gerar refutação por resolução não necessariamente em árvore, para uma fórmula F, ao contrário do DPLL. Refutação por resolução, o que o DPLL ou o DP fazem, é tentar reduzir a fórmula F para um absurdo, utilizando resolução[3]. As refutações geradas por DPLL

[3]No caso do DPLL implicitamente testando os dois ramos p, $\overline{p}$.

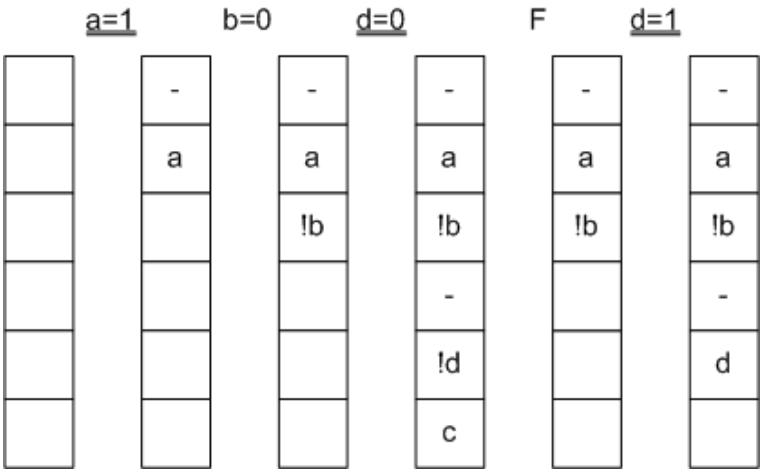


Figura 3.1. Pilha de Assinalamentos

ou DP são linearmente transformáveis em árvores de refutação e não em refutação generalizada, pois os resolventes no caso do DP e do DPLL não são reutilizados.

Uma refutação geral por resolução de uma fórmula na FNC F, é uma série S de cláusulas $\{C_1, C_2, \ldots, C_n\}$, tal que C_n é vazio, e ou cada C_k é copiado de F ou é a resolução de um par de cláusulas $\{C_i, C_j\}$, em que $i < j < k$. Em uma árvore de refutação, cada cláusula C_k só podem aparecer uma única vez. A refutação generalizada é mais poderosa que a árvore de refutação, uma vez que não limita a utilização dos resolventes[4]. Os algoritmos atuais conseguem gerar provas não necessariamente em árvore devido ao acréscimo de cláusulas de conflito, reinícios e o reuso de resolventes. Ao invés de exaustivamente fazer *backtracking* para achar a árvore de refutação como o DPLL, tenta-se acumular cláusulas de conflito suficientes para produzir uma refutação em menos tempo[5].

3.2 GRASP

O GRASP é um resolvedor de SAT proposto em [Marques-Silva & Sakallah, 1999]. Uma característica interessante deste artigo é que o Algoritmo 3.3 proposto pelo GRASP, e reapresentado em [Marques-Silva & Sakallah, 2000] mostra de forma procedural os módulos dos principais resolvedores de SAT atuais.

Será explicado o funcionamento geral apresentado no Algoritmo 3.3. A cada ramo da busca, um assinalamento de variável no nível d é selecionado pela função

[4]Existem fórmulas insatisfazíveis de tamanho n tal que a refutação em árvore é exponencial em n e a menor refutação geral é polinomial em n [Ben-Sasson et al., 2004].

[5]As cláusulas, e os conflitos derivados das cláusulas, podem participar zero ou mais vezes na geração de outros conflitos.

Algoritmo 3.3 SAT(d, β) por [Marques-Silva & Sakallah, 2000]

Entrada: in d - Nível corrente de decisão
Entrada: out β - Nível de decisão de backtrack
Saída: SATISFAZIVEL ou INSATISFAZIVEL

```
1: se Decide(d) != DECISAO então
2:    retorne SATISFAZIVEL;
3: fim se
4: enquanto true faça
5:    se Deduz(d) != CONFLITO então
6:       se SAT(d + 1, β) == SATISFAZIVEL então
7:          retorne SATISFAZIVEL
8:       senão se β != d || d == 0 então
9:          APAGA(d) {Backtracking não cronológico, volta a pesquisa para o nível
            d.}
10:         retorne INSATISFAZIVEL
11:      fim se
12:   fim se
13:   se Diagnostico(d, β) == CONFLITO então
14:      {inclui uma cláusula de conflito na base de dados e retorna o nível de back-
         tracking β}
15:      retorne INSATISFAZIVEL
16:   fim se
17: fim enquanto
```

Decide(d). O BCP é executado pela função *Deduz(d)*. Se não há conflito, então chama recursivamente a função SAT para o nível seguinte. Se a chamada é insatisfazível, então chama *Diagnostico*, que inclui uma cláusula de conflito na base de cláusulas de conflito e calcula o nível de *backtracking* retornado-o em β.

3.2.1 Aprendizagem de Cláusulas de Conflito e *Backtracking* Não Cronológico

O GRASP possui dois métodos bastante utilizados: aprendizagem de cláusulas de conflito, e por consequência, *backtracking* não cronológico. Novamente, a aprendizagem por cláusulas de conflito é um dos motivos, segundo Gomes et al [Gomes et al., 2008], responsáveis por levar os resolvedores SAT atuais em direção a refutação geral, que é considerado como fator principal da eficiência desses resolvedores em problemas reais. O conceito de cláusulas de conflito é recorrente nos resolvedores de SAT, como Chaff, Seção 3.4, e BerkMin, Seção 3.5, e representa a adição de cláusulas derivadas de um conflito à base de cláusulas. Pretende-se com essa adição não repetir os assinalamentos que levaram ao conflito. As cláusulas aprendidas são implicações das cláusulas originais

e podem ser adicionais e retiradas sem modificação na satisfabilidade do problema[6].

3.2.1.1 Grafo de Implicações

O GRASP contribui reformalizando o conceito de grafo de implicações. A partir do grafo, os métodos de geração de cláusulas de conflito e cálculo do nível de *backtracking* sobre as cláusulas de conflito são descritos em equações matemáticas (Equações 3.2, 3.3, 3.4, 3.5 e 3.6).

Há pelo menos duas formas de construção do grafo de implicações. O método do GRASP constrói o grafo de forma iterativa. Por outro lado, o método do zChaff reconstrói o grafo somente quando ocorre um conflito. Neste último caso, o assinalamento da variável pelo BCP sempre guarda a cláusula responsável pelo assinalamento, possibilitando a reconstrução, já que a cláusula responsável pelo BCP é justamente o conjunto de variáveis antecedentes[7].

O Algoritmo 3.4 mostra a construção do grafo de implicações pelo GRASP, que ocorre durante as funções *Decide*, *Deduz* e *Diagnostico*. Conforme a descrição desse algoritmo, a construção do grafo é feita durante o processo de BCP. Para cada BCP, adiciona-se uma ou mais arestas das variáveis da cláusula para o novo assinalamento. Durante o *backtrack*, removem-se as arestas relacionadas às cláusulas utilizadas em BCPs até o nível β. Seja o assinalamento de uma variável x através de BCP no nível d implicado pela cláusula w. Os assinalamentos antecedentes de x são denotados por $A^w(x)$, e representam os vértices das arestas que apontam para x. Estes vértices são todas as outras variáveis que não x de w, assinaladas antes de x e que segundo o BCP, levam ao assinalamento da variável x. Ao se desfazer o nível d, as arestas que contenham w são retiradas do grafo.

Para explicações da construção do grafo de implicações considere a fórmula proposicional na FNC dada em Exemplo 3.2. Esse exemplo é uma modificação do grafo de implicações retirado de [Zhang et al., 2001]. Nas figuras de grafos apresentadas nesta dissertação, o literal positivo, v, é determinado pela variável sem sinal v, e o negativo pelo prefixo $-$, $-v$, ou pelo literal barrado $\overline{v}$. Os grafos de implicações foram reduzidos e mostram somente os assinalamentos diretamente relacionados ao conflito, ou ao objetivo pretendido. O nível de assinalamento da variável é representado em parênteses. A legenda indica quais são as variáveis implicadas (azul claro), as variáveis de decisão (preto), evidenciando-se a variável de decisão do nível corrente (vermelho) e o conflito

[6] A base de cláusulas é divida em duas de acordo com o tipo da cláusula. A primeira é a base original do problema, a segunda é a base de cláusulas aprendidas ou base de cláusulas de conflito.

[7] A reconstrução é representada pela recursão da fórmula *causade* (Equação 3.3). Mais explicações na seção 3.4.3.

em marrom. As cláusulas que compõe as implicações são apresentadas nos vértices do grafo.

Exemplo 3.2 Fórmula proposicional utilizada nos grafos de implicações [Zhang et al., 2001].

$$
\begin{aligned}
w_1 &= V6 \vee \overline{V12} \vee \overline{V11} \\
w_2 &= V13 \vee V16 \vee \overline{V11} \\
w_3 &= \overline{V2} \vee V12 \vee \overline{V16} \\
w_4 &= V2 \vee \overline{V4} \vee \overline{V10} \\
w_5 &= V1 \vee \overline{V8} \vee V10 \\
w_6 &= V10 \vee V3 \\
w_7 &= V10 \vee V5 \\
w_8 &= \overline{V3} \vee \overline{V18} \vee \overline{V19} \\
w_9 &= V17 \vee \overline{V1} \vee \overline{V3} \vee \overline{V5} \vee V18 \\
w_{12} &= \overline{V10} \vee V17 \vee V30 \\
&\vdots \\
&\vdots
\end{aligned}
$$

No Exemplo 3.2, no nível 5, considere que o valor 0 foi atribuído à variável V17 no nível 3 e o valor 1 foi atribuído à variável V10 no nível 5. Um vértice para a variável V30=1 devido a w_{12}, no nível 5 é adicionado ao grafo. Essa parte do grafo é mostrada na Figura 3.2.

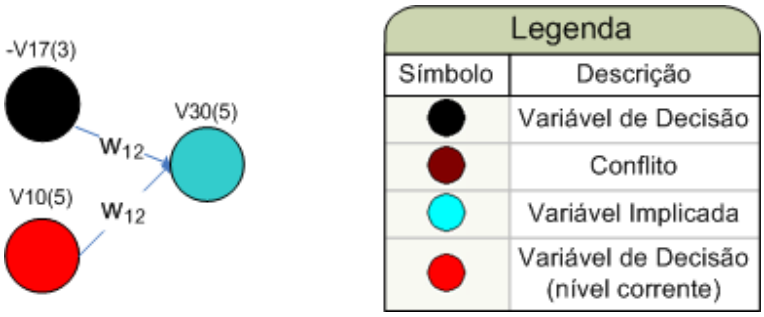


Figura 3.2. Um Grafo de Implicações para o Exemplo 3.2 [Zhang et al., 2001]

Na Equação 3.3, sejam dois conjuntos de nós em $A^w(x)$: O conjunto de nós assinalado no nível atual [8] e os nós assinalados antes do nível atual [9]. A^{w_C} corresponde

[8] $\sum(x) = \{(y, \mathcal{V}(y)) \in A^w(x) | \delta(y) = \delta(x)\}$
[9] $\Lambda(x) = \{(y, \mathcal{V}(y) \in A^w(x) | \delta(y) < \delta(x)\}$

aos assinalamentos associados ao conflito e w_C à cláusula de conflito.

Algoritmo 3.4 Construção do grafo de implicações pelo GRASP [Marques-Silva & Sakallah, 1999]

1. Cada vértice de I corresponde a um assinalamento de variáveis $x = \mathcal{V}(x)$, $\mathcal{V} = 0, 1$.
2. Os predecessores do vértice $x = \mathcal{V}(x)$ são os assinalamentos antecedentes $A^w(x)$, correspondente à cláusula unitária w que leva a implicação de x (BCP). As arestas direcionadas de $A^w(x)$ para o vértice $x = \mathcal{V}(x)$ são marcadas com w. Os vértices sem antecessores são decisões.
3. Vértices especiais de conflito são adicionados ao grafo para indicar a ocorrência de conflitos. Os predecessores deste vértice correspondem ao assinalamento de variáveis que fazem a cláusula w ficar insatisfazível e são vistos como antecedentes de $A^w(conflito)$. As arestas dos vértices de $A^w(conflito)$ são marcadas com w.

Equação 3.2 Nós Anteriores $A^{w_c}(k)$ ao conflito k [Marques-Silva & Sakallah, 1999]

$$A^{w_c}(k) = causade(k)$$

Equação 3.3 causade [Marques-Silva & Sakallah, 1999]

$$\begin{array}{ll} causade(x) = & \\ se(A^w(x) = \emptyset) & (x, \mathcal{V}(x)) \\ casocontrrio & \Lambda(x) \cup \left[\bigcup_{(y,\mathcal{V}(y))\in\sum(x)} causade(y)\right] \end{array}$$

A cláusula de conflito, que é adicionada na base e que determina o nível do *backtracking* não cronológico, e é calcula utilizando-se a Equação 3.4. Quando ocorre um conflito durante o BCP, o grafo é analisado para determinar aqueles assinalamentos que são responsáveis pelo conflito. A chamada à função descrita na Equação 3.3 dando como entrada o conflito, promove uma busca recursiva sobre os antecedentes do conflito até os assinalamentos de decisão. A conjunção dos assinalamentos de decisão encontrados na busca é um implicante, isto é , uma condição suficiente para que o conflito seja identificado ($\bigwedge literais \rightarrow conflito$). A negação da conjunção é uma implicação da função booleana original, que não existia na base anteriormente (Equação 3.4). Uma simples manipulação lógica prova que $\bigwedge literais \rightarrow conflito$, que foi derivado

Equação 3.4 Cláusula de Conflito [Marques-Silva & Sakallah, 1999]

$$w_C(conflito) = \sum_{(x,\mathcal{V}(x)) \in A_{w_C}(conflito)} x^{\mathcal{V}(x)}$$

$$Onde:$$

$$x^{\mathcal{V}}(x) = se(\mathcal{V}(x) = 0) \quad x$$

$$seno \quad \overline{x}$$

Equação 3.5 Calculo de β - nível de retorno do *backtracking* não cronológico [Marques-Silva & Sakallah, 1999]

$$\beta = max\{\delta(x) | (x, \mathcal{V}(x)) \in causade(conflito)\}$$

Equação 3.6 Cálculo do nível do assinalamento [Marques-Silva & Sakallah, 1999]

$$\delta(x) = max\{\delta(y) | (y, \mathcal{V}(y)) \in A^w(x)\}$$

da fórmula original, é logicamente equivalente a $\bigwedge literais_1 \rightarrow \bigvee \neg literais_2$, tal que $literais_1 \bigcup literais_2 = literais$. Logo a cláusula de conflito pode ser adicionada sem prejuízo a satisfabilidade[10]. O nível de *backtracking* não cronológico é o nível máximo das variáveis da cláusula de conflito exceto o nível atual. Esse nível é não incluído, isto é, a busca volta até o nível, sem incluí-lo.

Além dessas fórmulas, uma outra forma de explicar a geração de cláusulas de conflito é entendê-la como um corte no grafo de implicações, conforme explicado no artigo do zChaff[Zhang et al., 2001] (será explicado na Seção 3.4.3). A cláusula de conflito é gerada pela bipartição do grafo de implicações[11]. Neste caso, de um lado, estão as variáveis de decisão (lado da causa) e, de outro, as variáveis de conflito (lado do conflito). A cláusula de conflito é a disjunção de todos $y^{v(y)}$ para cada variável $(y, \mathcal{V}(y))$ que tem uma aresta do lado da causa para o lado do conflito, sendo que se $v(y)$=1 então $\overline{v}$, senão v. Dessa forma há muitas cláusulas de conflito passíveis de inclusão, tantas quantos forem os cortes.

[10] Prova mais formal desta afirmação encontra-se no artigo do GRASP.

[11] Podem ocorrer cortes que não envolvem o conflito e não biparticionam o grafo. Para informação consulte zChaff[Zhang et al., 2001].

3.2.1.2 UIPs

Melhorias no processo de diagnóstico do conflito envolvem a análise do grafo de implicações procurando implicantes mais fortes (cláusulas de conflito com menos literais). Estes implicantes são baseados nos UIPs (Unique Implicantion Points). Cada UIP pode ser visto como um gatilho para uma seqüência de assinalamentos no nível d, que geraria o mesmo conflito, porém com menos literais e conseqüentemente menos BCPs[12].

UIP, por definição, é um nó no nível de decisão corrente, tal que todo caminho entre a variável de decisão e o conflito passa pelo UIP[13]. Para que a cláusula de conflito possua menos variáveis, pode-se modificar a fórmula *causade* para parar em quaisquer UIPs e não somente sobre o UIP de decisão[14].

Os UIPs podem ser identificados através da busca em largura a partir do conflito. Se é identificado apenas um nó do nível de decisão durante a busca, então este é um UIP. Considere a Figura 3.3. Os vértices em preto e vermelho representam as variáveis de decisão. Os conceitos de 1-UIP e Todos-UIP será mostrado no Seção 3.4.3. O vértice -V10(7) é um UIP. Percebe-se que este vértice domina as variáveis de decisão V11, V13, V6 e V4, já que ele é implicada pela conjunção destas variáveis. Os vértices interiores do grafo também são vértices no nível corrente, passíveis de serem UIPs, basta que todos os caminhos da variável de decisão do nível corrente, para o conflito, passem por ele. Logo, os UIPs desse grafo são: V11, V2 e V10.

Na detecção do conflito, a função *Diagnostico*[15] pode gerar diversas cláusulas de conflito (utilizando diversos UIPs). Dessa forma, com a Equação 3.4 e com os UIPs pode-se derivar as cláusulas contidas em Enumeração 3.1.

Enumeração 3.1 Os UIPs do grafo de implicações Figura 3.3.

1. UIP V11 $\rightarrow V6 \vee \overline{V11} \vee V13 \vee \overline{V4} \vee \overline{V8} \vee \overline{V19} \vee V17$
2. UIP -V2 $\rightarrow V2 \vee \overline{V4} \vee \overline{V8} \vee \overline{V19} \vee V17$
3. UIP -V10 $\rightarrow V10 \vee \overline{V19} \vee \overline{V8} \vee V17$

O *backtracking* não cronológico, por sua vez, envia a busca para o nível mais baixo (maior) das variáveis que levaram ao conflito (Equação 3.5). Retira-se dessa fórmula o nível corrente. Como demonstrado no artigo do GRASP, qualquer volta a um nível

[12] São mais fortes porque os cortes UIPs são dominantes e fornecem cláusulas menores, segundo o GRASP [Marques-Silva & Sakallah, 1999].

[13] Este conceito é estendido na seção 3.4.3.

[14] O literal de decisão do nível corrente é por definição um UIP

[15] Veja o Algoritmo 3.3.

inferior ao determinado na busca continua não satisfazendo a fórmula. Os níveis de retorno para cláusulas de conflito são determinados na Enumeração 3.2.

Enumeração 3.2 Os níveis até onde o *backtracking* não cronológico deve ser realizado para os UIPs da Enumeração 3.1.

- UIP V11 → 6
- UIP -V2 → 6
- UIP -V10 → 5

Uma característica interessante da cláusula de conflito que possui UIPs é a troca de valor da variável do UIP. Seja a cláusula de conflito do UIP V11. O valor de V11 na cláusula é o inverso do valor da variável de decisão V11. O mesmo ocorre com os UIPs -V2 e -V10. Devido a essa característica, o zChaff chama esse modo de aprendizagem de *flip mode*. As variáveis dos UIPs passam a determinar seus valores em níveis anteriores de backtracking, com o valor do literal invertido. Após o retorno ao nível determinado pelo conflito, o valor do UIP deve ser escolhido como o próximo valor a ser atribuído[16]. Quando o UIP não é a variável de decisão do nível corrente, mesmo o valor da variável sendo determinado pelo BCP, o zChaff se refere aos UIPs como *fake decisions*[17].

3.3 Aplicação de Heurísticas

A divisão demonstrada no Algoritmo 3.3, indica em que módulos as heurísticas podem ser aplicadas, sendo eles *Deduz*, *Diagnostico* e *Decide*. A fase Deduz é a fase do BCP. Heurísticas e estruturas de dados utilizadas nessa fase são descritas em [de Campos Lynce Faria, 2004] e incluem contadores (GRASP), listas encadeadas, e *Two-Watched Literals*. A fase *Diagnostico* inclui o *backtracking* não cronológico, o *backtracking* cronológico e o aprendizado de cláusulas de conflito. A fase *Decide* é a fase onde a ordem das variáveis de decisão são escolhidas e inclui: MOM (*Maximum Occurrences on clauses of Minimum sizes*); e a VSIDS (*Variable State Independent Decaying Sum* proposto por [Malik et al., 2001]). A VSIDS será explicada posteriormente na Seção 3.4. A heurística MOM atribui valor à variável que aparece no maior número de cláusulas de menor tamanho. Dessa forma, mais variáveis serão assinaladas pelo

[16] Já que as outras variáveis da cláusula de conflito mantiveram seu valor, o BCP deve ser ativado.

[17] Possivelmente pelo fato de o valor dessa variável passar a ser atribuído em níveis anteriores da busca, ela não ser inicialmente uma variável de decisão e passar a influenciar o valor da antiga variável de decisão.

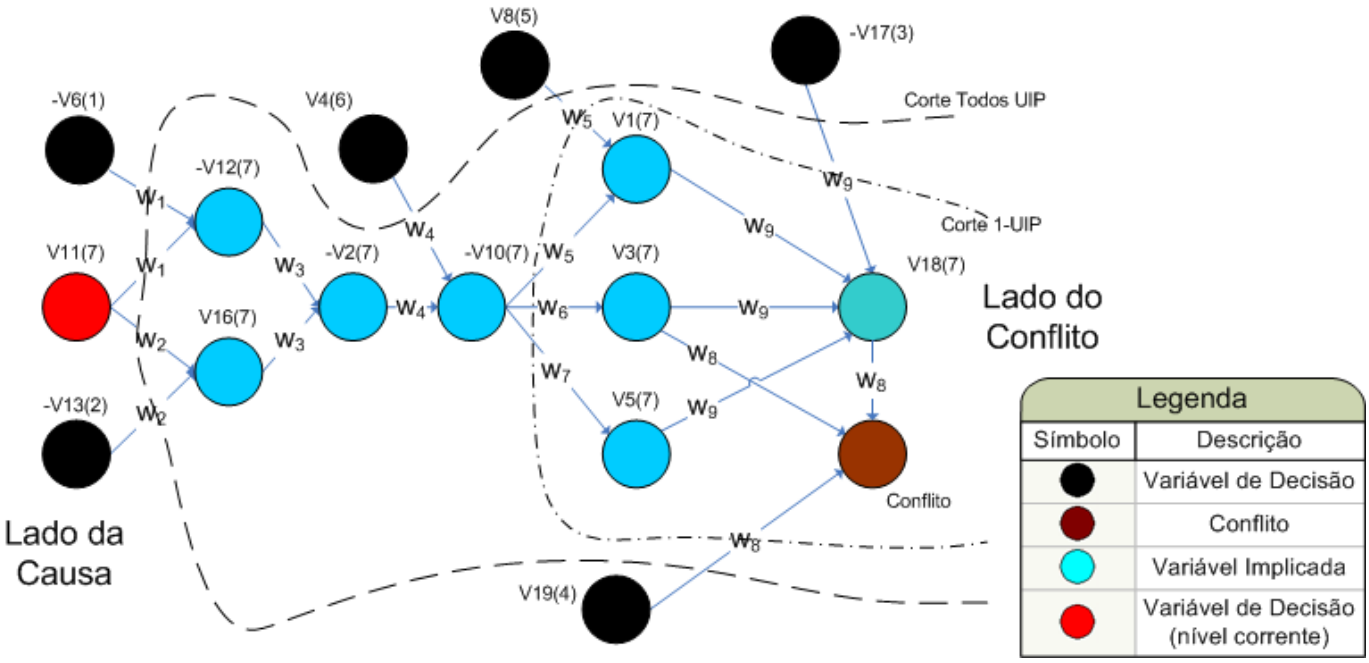


Figura 3.3. Um Grafo de Implicações para o Exemplo 3.2 [Zhang et al., 2001]

BCP. [Ryan, 2002] também aponta algumas heurísticas de *Deduz*, em que as cláusulas binárias e ternárias tem uma estrutura própria.

3.3.1 Heurísticas em Outros Módulos

Conforme apontado por [Ryan, 2002], o BCP opera de forma ampla e não seqüencial sobre o conjunto de cláusulas, que é muitas vezes maior que a *cache* L2 de um computador. Essa forma difusa de acesso implica em um número alto de acesso à memória e de *cache misses* que gera a degradação de tempo do BCP e, por consequência, sua dominação sobre as outras partes do resolvedor.

A remoção de cláusulas aprendidas ajuda na diminuição do consumo de memória bem como na diminuição da quantidade de *cache misses* que os acessos produzem. Heurísticas de eliminação de cláusulas aprendidas como a apontada por Chaff na Seção 3.4 tentam diminuir o conjunto de cláusulas e o tempo de execução.

Outra técnica bastante utilizada é o reinício da busca. Segundo alguma condição, por exemplo um limite x de memória foi atingido, a busca é reiniciada, voltando-se ao nível 0 de decisão, contendo além da base inicial, as cláusulas já aprendidas. Essa técnica permite escolher um novo ramo de busca, desde a raiz. Isso pode enviar a busca para ramos mais promissores uma vez que as cláusulas aprendidas podem indicar uma nova ordem nas buscas.

Com estas modificações, apresenta-se o Algoritmo 3.5, uma modificação do Algoritmo 3.3 que leva em conta tais heurísticas. Uma breve explicação de cada heurística é mostrada na Enumeração 3.3. Durante o texto, o nome da heurística poderá denominar a heurística.

Enumeração 3.3 Explicação de cada heurística do resolvedor mostrado no Algoritmo 3.5

- ReduceDB - eliminação das cláusulas aprendidas. O método em si é chamado de DBManag, do inglês *Database Management*. A condição na qual o método é chamado, foi denominada ReduceDBCond.
- Decide - seleciona a próxima variável e seu valor. Esta variável é conhecida como variável de decisão.
- Deduz - geralmente utiliza-se o BCP. Determina os valores de outras variáveis a partir das cláusulas que se tornaram unitárias devido as decisões anteriores.
- Diagnostico - aprendizagem de cláusulas de conflito e *backtracking* não cronológico;
- Reinício - reinícia a busca no nível zero mantendo as cláusulas aprendidas. Possui uma condição (*condicao*2), na qual o método de reinício é chamado.

3.4 SATO, Chaff e zChaff

Como enfatizado pelos autores do resolvedor Chaff [Malik et al., 2001], tipicamente noventa por cento do tempo de execução de um resolvedor DLL com aprendizado de cláusulas é gasto no BCP. A estrutura de dados utilizada pelos resolvedores influi fortemente no tempo gasto no BCP. O GRASP utiliza métodos baseados em contagem. Já o Chaff utiliza o algoritmo chamado *Two-Watched Literals*. Dada a sua importância e a utilização no resolvedor proposto, vai-se descrever o funcionamento básico desse algoritmo[18].

3.4.1 *Two-Watched Literals*

Para se implementar BCP de forma eficiente, é necessário visitar apenas as cláusulas que se tornaram recentemente implicadas pela adição de um conjunto de assinalamentos. As cláusulas são ditas implicadas, se todos os literais exceto um são assinalados em

[18]Mais uma vez para uma revisão bastante ampla das estruturas de dados consulte [de Campos Lynce Faria, 2004].

Algoritmo 3.5 modSAT(d, β)

Entrada: in d - Nível corrente de decisão
Entrada: out β - Nível de decisão de backtrack
Saída: SATISFAZIVEL ou INSATISFAZIVEL

```
1: se Decide(d) != DECISAO então
2:   retorne SATISFAZIVEL;
3: fim se
4: enquanto true faça
5:   se Deduz(d) != CONFLITO então
6:     se SAT(d + 1, β) == SATISFAZIVEL então
7:       retorne SATISFAZIVEL
8:     senão se β != d || d == 0 então
9:       APAGA(d) {Backtracking não cronológico, volta a pesquisa para o nível
         d.}
10:      retorne INSATISFAZIVEL
11:    fim se
12:  fim se
13:  se Diagnostico(d, β) == CONFLITO então
14:    {inclui uma cláusula de conflito na base de dados e retorna o nível de back-
       tracking β}
15:    retorne INSATISFAZIVEL
16:  fim se
17:  {ReduceDB inclui condição e o método}
18:  se condicao então
19:    {Exemplo tamanho da base > 100MB}
20:    DBManag()
21:  fim se
22:  se condicao2 então
23:    {Exemplo 1000 conflitos gerados}
24:    Reinício()
25:  fim se
26: fim enquanto
```

zero. Uma maneira natural seria verificar primeiramente as cláusulas que tem o literal decido em *Decide* assinalado em zero[19], e posteriormente verificar as cláusulas em que o BCP assinala um de seus literais para zero. Pode-se, também, manter um contador que indica quantos literais da cláusula tem valor 0. Dessa forma não é necessário visitar as cláusulas enquanto o contador não passe de $N-2$ para $N-1$, sendo N o número de literais da cláusula.

Uma aproximação desse método é feita no algoritmo *Two-Watched Literals*. Pega-se quaisquer dois literais de cada cláusula não assinalados em zero, para vigiar seu valor. Garante-se, desta forma, que até que um destes literais seja assinalado para zero, não pode haver mais de $N-2$ literais assinalados para zero, isto é, a cláusula não é implicada. Agora visita-se cada cláusula quando um dos seus literais vigiados é assinalado para zero. Ocorrem dois casos escritos na Enumeração 3.4. Um benefício desta técnica é que o *backtracking* não necessita de nenhuma alteração nos literais vigiados. Reduz-se assim o número de acessos à memória e, por consequência, diminuise o número de *cache misses*, que forma o principal gargalo dos resolvedores de SAT. O resolvedor SATO, [Zhang, 1997], foi o primeiro a propor esta técnica. Porém havia uma direção fixa da busca dos literais vigiados, dos extremos para o centro, o que inclui custo adicional durante o *backtracking* (os literais vigiados ponta inicial e a ponta final deveriam voltar para os literais não assinalados mais para o início ou mais para o fim, respectivamente).

Enumeração 3.4 Casos a serem verificados quando o literal vigiado é assinalado para zero.

(1) A cláusula possui outro literal não assinalado com zero, além do outro literal vigiado. Nesse caso, o literal vigiado assinalado é trocado por aquele literal.

(2) A cláusula é implicada. Aplica-se BCP sobre o outro literal vigiado.

A figura 3.4 mostra o BCP com a estrutura de dados *Two-Watched Literals*. Após a resolução do conflito, os literais que eram vigiados antes do conflito continuam os mesmos (V2 e -V4).

3.4.2 Heurísticas da Função *Decide*

O resolvedor SATO escolhe a variável x tal que o valor de $f_2(x) * f_2(\overline{x})$ é máximo, onde $f_2(L)$ é o número de ocorrências do literal L nas cláusulas de tamanho 2, dentro do conjunto de $[a * n]$ cláusulas positivas. Cláusulas positivas são as cláusulas onde todos

[19] Lembre-se da regra I do DP, que retira os literais negados das cláusulas.

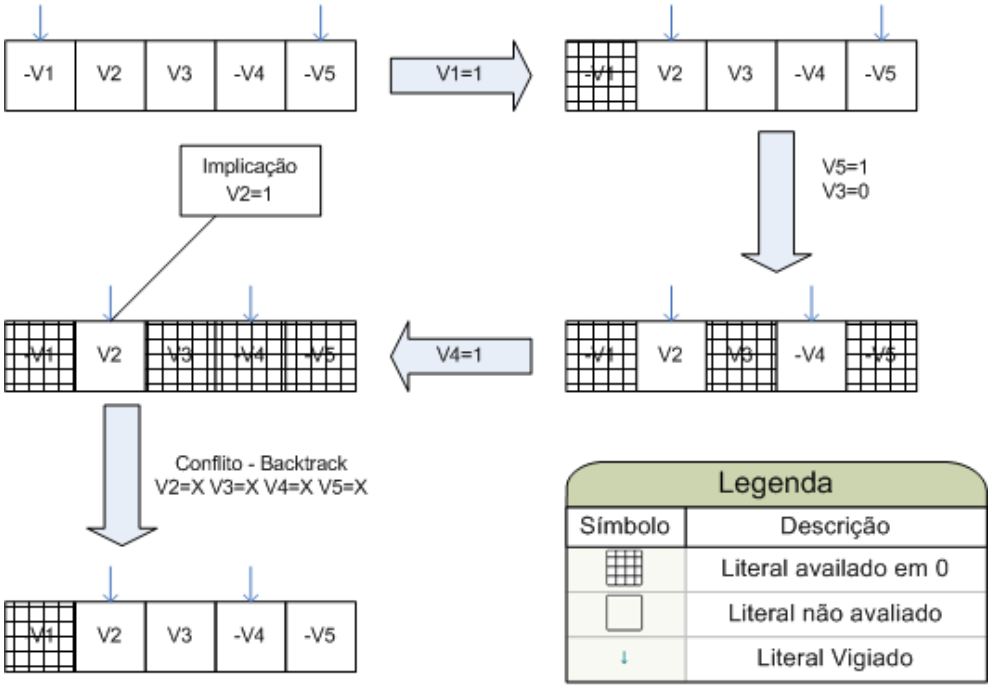


Figura 3.4. BCP com "Two-Watched Literals"

os literais são positivos. Estas heurísticas são relativas aos problemas Quasigroup, como o problema "Latin square".

Algoritmo 3.6 VSIDS [Malik et al., 2001]

---

(1) Cada variável v em cada polaridade (v, $\overline{v}$) tem um contador iniciado em zero.

(2) A cada cláusula de conflito adicionada à base, o contador de cada literal é incrementado.

(3) A variável não assinalada e polaridade com o maior contador é a variável de decisão.

(4) Empates são resolvidos aleatoriamente.

(5) Periodicamente todos contadores são divididos por uma constante.

O Chaff utiliza o VSIDS, descrito no Algoritmo 3.6. A divisão dos contadores cria a tendência de busca pela satisfabilidade das cláusulas mais recentemente adicionadas. Os autores avaliaram diversas heurísticas de *Decide* e para comparação preferiram a utilização do tempo de execução à quantidade de decisões. A argumentação é que o número de decisões menor pode indicar uma heurística mais inteligente, porém o custo de utilização da heurística pode torná-la menos interessante que outra que tem mais

decisões, mas resolve mais rápido. A VSIDS teve os melhores resultados, comparadas a outras implementadas, mas não divulgadas no código-fonte, como DLIS ou DLCS.[20]

3.4.3 Estudo de Aprendizagem pelo zChaff

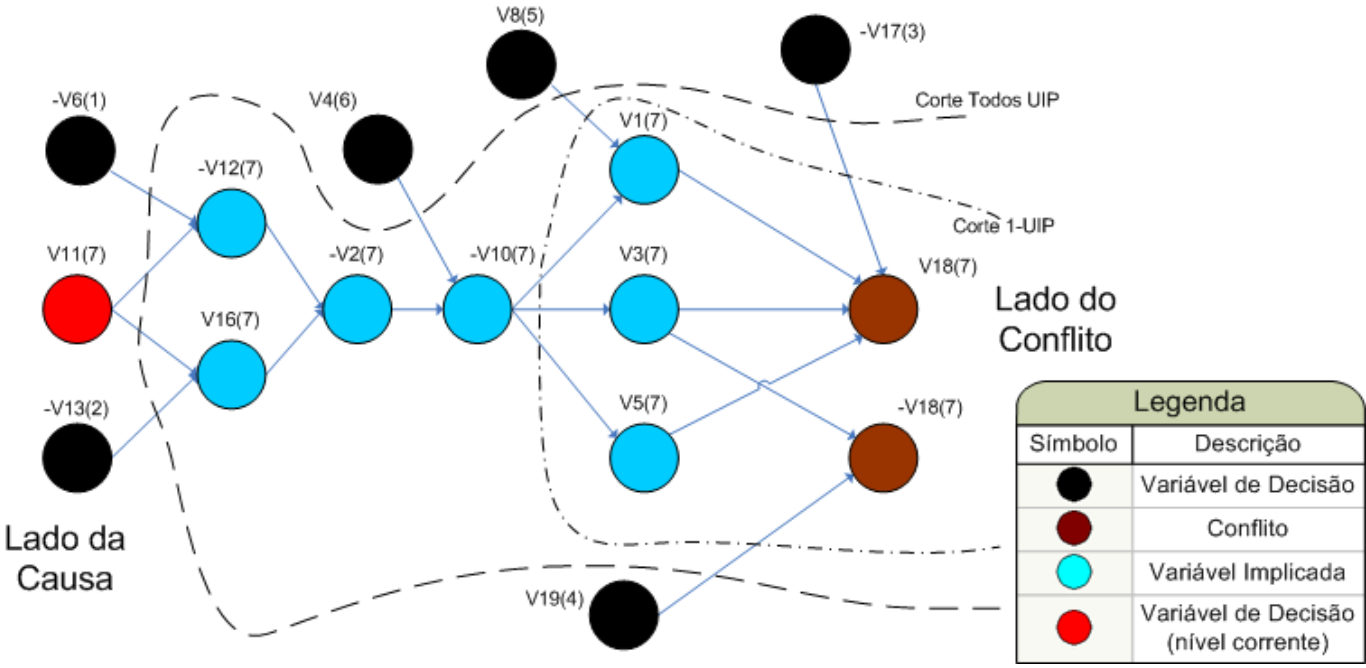


Figura 3.5. Grafo de implicações do Exemplo 3.2

O artigo [Zhang et al., 2001] mostra um estudo sobre os método de aprendizagem de cláusulas através do grafo de implicações Figura 3.5, principalmente do Todos-UIP e do 1-UIP. O conceito de UIP pode ser estendido para o níveis diferentes do corrente. O 1-UIP corresponde ao Primeiro-UIP do nível corrente. Primeiro-UIP indica o primeiro UIP encontrado no nível n na busca a partir do conflito. O 2-UIP requer que o Primeiro-UIP do nível imediatamente superior da busca também esteja do lado da causa. O Todos-UIP requer que todos Primeiros-UIPs de todos os níveis até o nível mais superior estejam do lado da causa (todas são variáveis de decisão).

A cláusula de conflito é construída como um BCP no sentido inverso, através de resolução. As cláusulas responsáveis pelo assinalamento da variável são resolvidas até que contenha apenas uma variável que foi assinalada no nível corrente de decisão. Neste artigo, o grafo para o 1-UIP é construído a partir do conflito, da mesma forma

[20] Veja mais sobre as heurísticas testadas em [Malik et al., 2001].

que é feita no Minisat e no BerkMin. A construção do grafo apresentado na Figura 3.5, para o 1-UIP será apresentada.

O algoritmo de construção do grafo e obtenção da cláusula de conflito para o 1-UIP é retirado de [Ryan, 2002]. Considere como entrada do algoritmo a Enumeração 3.5. O Algoritmo 3.7 utiliza algumas variáveis inteiras na busca. i é o índice da cláusula sendo resolvida, y é o nível de decisão mais recente (conflito) e é constante após R0. m conta as variáveis do nível y que estão marcadas mas que não foram resolvidas ou incluídas na cláusula de conflito, a. p é um apontador para a pilha de assinalamento.

Enumeração 3.5 Entrada para o algoritmo de construção do grafo através das cláusulas BCP

- Fórmula F na CNF (F_i indica a cláusula i-ésima).
- Pilha de assinalamentos S (S_i indica o i-ésimo elemento da pilha. S_0 é o inicial, e S_T o mais recente).
- Conjunto de *Flags* das variáveis U (Para cada variável há um *flag* U_v. Inicialmente se v é assinalado no nível 0 então U_v é verdadeiro. Senão é falso).
- Função antecedente C (C_v é o índice da cláusula que se tornou unitária e foi utilizada no BCP para assinalamento da variável v. É indefinido para variáveis de decisão).
- Função nível L (L_v é o nível de decisão em que a variável foi assinalada. É indefinido para variáveis livres.).
- X é o índice da cláusula geradora do conflito (F_X é a cláusula que gerou o conflito, que possui todos os literais falsos).

O Algoritmo 3.7 faz resolução entre os literais do nível de decisão corrente da busca e as cláusulas que assinalaram esses literais através do BCP ($i = C(v)$), e para quando há apenas um literal no nível corrente[21]. O valor deste literal deve ser trocado (pois este literal está do lado da causa no corte). Para construção do grafo Todos UIP basta modificar o Algoritmo 3.7 para Algoritmo 3.8. Seja nl a variável do literal de decisão do nível corrente. Neste caso, o algoritmo para somente se a única variável do nível corrente é a variável de decisão deste nível ($v = nl$).

[21] É bom lembrar que a cláusula de conflito possui todos os literais falsos. Logo ela é passível de resolução com as cláusulas que assinalaram os literais como verdadeiros. Isso ocorre ou por BCP ou por decisão. A condição de parada é a definição do UIP.

Algoritmo 3.7 Algoritmo para construção do grafo de implicações para o 1-UIP[Ryan, 2002].

R0. Assinala $a = []$, $i = X$, $m = 0$, e $p = T$. Assinala $y = L(v)$, em que v é uma variável de S_T.

R1. Para cada literal l de F_i:

R1a. Seja v a variável de l. Se $U_v = V$, pula R1b.

R1b. Assinala $U_v = V$. Se $L(v) < Y$, coloque l em a. Senão, assinala $m = m + 1$.

R2. Seja v a variável de S_p. Se $U_v = V$, vá para R4.

R3. Assinala $p = p - 1$. Vá para R2.

R4. Se $m = 1$, coloca $\overline{S_p}$ em a e retorna a. Senão, assinala $p = p - 1$.

R5. Assinala $m = m - 1$. Assinala $i = C(v)$. Vá para R1.

Algoritmo 3.8 Algoritmo para construção do grafo de implicações para o Todos-UIP baseado no 1-UIP de [Ryan, 2002].

R0. Assinala $a = []$, $i = X$ e $p = T$. Assinala $y = L(v)$, em que v é uma variável de S_T.

R1. Para cada literal l de F_i:

R1a. Seja v a variável de l. Se $U_v = V$, pula R1b.

R1b. Assinala $U_v = V$. Se $L(v) < Y$, coloque l em a.

R2. Seja v a variável de S_p. Se $U_v = V$, vá para R4.

R3. Assinala $p = p - 1$. Vá para R2.

R4. Se $v = nl$, coloca $\overline{S_p}$ em a e retorna a. Senão, assinala $p = p - 1$.

R5. Assinala $i = C(v)$. Vá para R1.

3.4.4 Outras Heurísticas

O Chaff, como outros resolvedores, aplica a remoção de cláusulas aprendidas, *ReduceDB*. No caso do Chaff, utiliza-se remoção preguiçosa. A cláusula, ao ser incluída, determina em que ponto do futuro ela será removida. Este ponto é definido em uma métrica (no caso, número de literais não assinalados da cláusula entre 100-200) para remoção das cláusulas. Quando uma cláusula obedece à métrica, ela é marcada como removida. A memória ocupada pelas cláusulas removidas é posteriormente retomada para abrigar outras cláusulas. Isso só ocorre quando a cláusula marcada como removida não é responsável por nenhum assinalamento corrente.

3.5 BerkMin

O BerkMin se destacou nos testes de [Andrade, 2008] e, como dito anteriormente, o trabalho envolve, também, implementar as heurísticas do BerkMin, testá-las de forma modular e disponibilizá-las.

O resolvedor BerkMin, assim como o Chaff, utiliza a estrutura de dados *Two-Watched Literals* para BCP; e o conceito de redução da contribuição de cláusulas antigas no processo de decisão (mas não através da divisão dos contadores). Do GRASP, o BerkMin utiliza os procedimentos de análise de conflitos e *backtracking* não cronológico. O trabalho do BerkMin é encontrado em dois artigos [Goldberg & Novikov, 2002], [Goldberg & Novikov, 2007]. Utiliza também reinício, que corresponde a desfazer todos os assinalamentos e reiniciar a busca do zero, levando em consideração as cláusulas aprendidas.

Um diferencial que se observa no artigo [Goldberg & Novikov, 2007] são os testes comparativos entre o tempo de uso ou não das novas heurísticas propostas. Quando a heurística proposta não é aplicada utiliza-se uma heurística baseada na do Chaff ou na do GRASP. Nesse artigo também foram feitos testes com todas as heurísticas propostas habilitadas contra o zChaff. Os testes em [Goldberg & Novikov, 2002] são feitos entre o BerkMin e o Chaff. Em [Goldberg & Novikov, 2007], os resultados são melhores quando todas as heurísticas são ativadas e, em [Goldberg & Novikov, 2002], o BerkMin é melhor que o Chaff em quase todos os casos.

3.5.1 Novas Heurísticas

As heurísticas de decisão do BerkMin são baseadas em duas observações. Primeiro, um grande número de fórmulas na FNC de problemas reais são fáceis para resolvedores

como Chaff. Logo, essas instâncias possuem um pequeno subconjunto de variáveis que conseguem deduzir cláusulas pequenas, já que o BCP é mais rápido neste tipo de cláusulas. Segundo, o conjunto de variáveis que fazem parte das cláusulas de conflito podem mudar rapidamente. Os autores demonstram isto com uma porta AND. A Figura 3.6 mostra essa porta. Se uma entrada da porta AND é zero, então as cláusulas relativas as outras entradas não são relevantes, já que o resultado é sempre zero. Se a entrada passa a ser um, então as cláusulas das outras entradas passam a influenciar na busca da satisfabilidade. Na Figura 3.6, a mudança de valor da entrada para 1, modifica as cláusulas de conflito geradas, uma vez que a porção rachurada passa a ser relevante. Essas observações levaram a duas conclusões: o processo de decisão deve ser sensível o suficiente para permitir uma boa escolha de variáveis e deve dinamicamente mudar de um conjunto de variáveis para outro. A mudança dinâmica é feita em Chaff com a divisão dos contadores de cada literal. As mudanças que foram feitas pelo BerkMin estão descritas na Enumeração 3.6.

Enumeração 3.6 Novas heurísticas desenvolvidas pelo BerkMin [Goldberg & Novikov, 2002].

1. A sensibilidade da escolha da variável de decisão foi aumentada.
2. A mobilidade da escolha da variável de decisão foi aumentada.
3. Uma nova heurística de exame de qual valor será atribuído foi proposta. A heurística é chamada de Simetrização da Base.
4. Um novo procedimento de controle da base de cláusulas aprendidas foi introduzido.

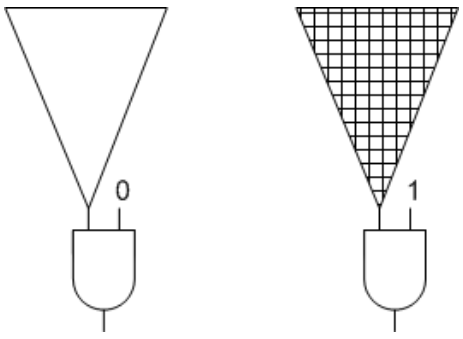


Figura 3.6. Mudança dos literais de Cláusula de Conflito

3.5.1.1 Sensibilidade da Escolha da Variável

Tanto o BerkMin quanto o Chaff possuem o conceito de atividade da variável ($var\_activity$). Porém, enquanto o Chaff incrementa a atividade da variável para cada ocorrência na cláusula de conflito final, o BerkMin incrementa o contador para todas as cláusulas que foram utilizadas durante a resolução do conflito. A divisão dos contadores é feita utilizando o fator 4 ao invés de 2 para o Chaff.

3.5.1.2 Mobilidade da Escolha da Variável

Essa seção aborda a escolha da variável de decisão. No BerkMin, as cláusulas de conflito são organizadas como uma pilha. Chama-se de cláusula corrente do topo aquela mais próxima do topo ainda não satisfeita. A próxima variável de decisão é escolhida entre as variáveis livres da cláusula corrente do topo. Tal variável é escolhida pelo maior valor de atividade. Se não há cláusula corrente do topo, então a variável, não atribuída, de maior atividade é escolhida.

3.5.1.3 Simetrização da Base

Após escolhida a variável deve-se decidir seu valor. Essa seção aborda a escolha do literal.

Para cada literal l, existe um contador de atividade denominado $lit\_activity$, que mostra quantas cláusulas de conflito foram geradas que continham o literal l. Este contador nunca é dividido, ao contrário de $var\_activity$. O valor é escolhido da seguinte forma: suponha que a variável escolhida foi v; o literal v, ou $\overline{v}$ que possuir o maior contador é escolhido. Se v for escolhido, então $v = 1$, senão $v = 0$. Com isso, há uma tendência de equiparação dos contadores, uma vez que a escolha de $v = 1$, implica que se a variável v aparecer na cláusula de conflito, então o literal é $\overline{v}$. Por outro lado, a escolha de $v = 0$ implica que se v aparecer será com o literal v.

O método de escolha do valor do literal, descrito anteriormente, é utilizado apenas se houver cláusulas de conflito não satisfeitas. Se não houver, então estamos satisfazendo cláusulas da base inicial. Nesse caso, uma heurística denominada $nb\_two(l)$, onde l é um literal, é utilizada. Essa heurística calcula aproximadamente quantas cláusulas binárias estão na vizinhança de l. Primeiro, conta-se o número de cláusulas binárias com l. Para cada cláusula binária que contém l, conta-se o número de cláusulas binárias de $\overline{v}$, onde v é o outro literal da cláusula binária. A soma desses contadores é nb_two. O literal com maior nb_two é escolhido.

Segundo os autores, a técnica de simetrização ajuda no tempo de execução quando ocorrem reinícios, como é o caso do resolvedor desenvolvido nesta dissertação.

3.5.1.4 Gerência das Cláusulas Aprendidas

Cada cláusula C possui um contador denominado $clause\_activity(C)$, que conta o número de cláusulas de conflito que C foi responsável. Cada cláusula possui também um tamanho, isto é, o número de literais, denominado $length(C)$. A base de cláusulas aprendidas é, na verdade, um fila, onde as novas cláusulas são incluídas no final da fila. Os $\frac{1}{16}$ iniciais são denominados cláusulas velhas e o restante, cláusulas novas. As cláusulas novas são mantidas se $length(C) < 43$ ou se a $clause\_activity(C) > 7$. As cláusulas velhas são mantidas se $length(C) < 9$ ou se a $clause\_activity(C) > threshold$. O $threshold$ é no início 60 e é incrementado a cada 1024 decisões. O reinício ocorre a cada 550 conflitos. A cada reinício o procedimento de remoção das cláusulas de conflito é iniciado. Se ao final menos de $\frac{1}{16}$ de cláusulas não são eliminadas, o tamanho das cláusulas velhas é decrementado até o limite de 4.

Percebe-se, com o exemplo da porta AND, que o aumento das sensibilidades da escolha e da mobilidade das variáveis podem ajudar muito na verificação de CEC.

O artigo a seguir é a respeito de uma técnica aplicada ao BerkMin para melhorar o desempenho do resolvedor para o problema de Equivalência de Circuitos Combinacionais Dissimilares.

3.6 Equivalence Checking of Dissimilar Circuits

Escrito pelos mesmos autores que o BerkMin, este artigo apresenta modificações que visam a integração ao resolvedor de SAT de aspectos estruturais do circuito original. Os métodos envolvem a utilização de métodos baseado em circuitos com uma especificação comum (CS, do inglês *Common Specification*) e da distância da variável proposicional que representa a saída de uma porta lógica e a entrada do circuito.

O artigo [Goldberg & Novikov, 2003] explica que o CS é uma forma de generalização da noção de similaridade estrutural. Os autores Goldberg e Novikov adaptaram os padrões mostrados por CEC com CS no resolvedor BerkMin. Acredita-se que estas adaptações sejam a estratégia 1 do BerkMin. Através das modificações dos procedimentos de decisão e reinício, os autores realizam um espelhamento do que eles chamam de provas curtas por resolução.

3.6.1 Decide

No procedimento de decisão, a escolha de variável ocorre da cláusula do topo com maior atividade, mesmo que ela já seja satisfeita. Uma segunda mudança no procedimento de decisão é a multiplicação durante a fase de ordenação das variáveis de uma função monotônica crescente, onde uma das variáveis livres seja o nível da variável. Por nível entende-se o nível topológico da variável começando das entradas.

3.6.2 Reinício

No procedimento de reinício, a mudança é a implementação de reinício forte e fraco. Os reinícios fortes ocorrem na profundidade maior que 15 no segundo conflito da busca. Reinícios fortes ocorrem a cada 550 conflitos como é feito pelo BerkMin. A diferença básica, entretanto, é que durante reinícios fortes ocorre a limpeza da base de cláusulas aprendidas. O argumento é que os reinícios freqüentes simulam as operações de correlação e filtragem, explicadas no artigo.

3.7 Minisat

Esta seção é dedicada ao resolvedor escolhido como base para as modificações propostas no BerkMin e teste da hipótese. Relembrando uma das hipóteses que desenvolvemos neste trabalho é que uma política mais agressiva de redução da base de cláusulas aprendidas deve melhorar o tempo de execução do resolvedor.

O artigo sobre o Minisat [Eén & Sörensson, 2003] foi escrito para a versão 1.2, anterior à adotada, porém a idéia geral de cada módulo foi mantida. O resolvedor Minisat foi construído para ser de fácil modificação. As modificações em um resolvedor SAT, ou sua implementação do zero, envolvem tanto um conhecimento das técnicas, como Aprendizado de Cláusulas de Conflito e *Two-Watched Literals*, quanto do domínio do problema que se deseja resolver. O resolvedor é pequeno, o que também facilita no aprendizado, e eficiente, sendo o ganhador na categoria industrial da competição SAT 2005.

Como o Minisat utiliza os métodos apresentados nas seções anteriores, preferiu-se utilizar uma tabela para apresentar a origem de cada método que o Minisat utiliza. Na Tabela vê-se a associação entre os principais métodos e as heurísticas apresentadas na Seção 3.3. No caso da heurística *Reincio*, o *Método* indica em que método da classe o reinício é chamado, enquanto nos outros casos indica efetivamente qual método implementa a heurística. Na heurística *Decide*, o VSIDS modificado faz o incremento de

atividade da variável ao invés do literal, como é feito no VSIDS original (veja Seção 3.4.2). A decisão da variável pode ser também randômica. A escolha do sinal da variável é determinada por parâmetros, e pode ser: negado (0); afirmado (1); randômico; ou definido pelo usuário. Observa-se que durante o procedimento de aprendizado da cláusula de conflito usando 1-UIP, que o Minisat tenta simplificar a cláusula de conflito retirando algum literal redundante. Mais explicações sobre o método ReduceDB do Minisat podem ser encontradas na seção 4.1.1.

Heurística	**Método**	**Base**
Decide	**pickBranchLit**	*VSIDS* modificado
Deduz	**propagate**	*BCP - "Two-Watched Literals"*
Diagnostico	**analyze**	Primeiro *"UIP"* - BCP reverso por resolução[22]
ReduceDB	**reduceDB**	Redução utilizando atividade da cláusula.[23]
Reinício	**search**	Número de conflitos chega ao limite estabelecido.

Tabela 3.1. Tabela com a Relação Heurística Módulo do Minisat.

Capítulo 4

Resolvedor Modular de Satisfabilidade Aplicado na Verificação de Circuitos Combinacionais

Este capítulo descreve as modificações realizadas, de forma modular no código do Minisat, versão 2.0 beta, feitas conforme entendimento dos artigos do BerkMin ([Goldberg & Novikov, 2002], [Goldberg & Novikov, 2007]) e do artigo [Goldberg & Novikov, 2003]. Quatro esquemas de implementação dos módulos foram analisados: herança direta, pré-processamento C, Programação Orientada por Aspectos, e pré-processamento próprio baseado em XML. Uma metodologia de integração e modificação dos módulos foi proposta, e visa torná-los o mais independentes possível, de forma a torná-los intercambiáveis. Esta metodologia, em alguns casos, inclui a replicação de algumas estruturas de dados.

4.1 Introdução

Freqüentemente novos resolvedores de SAT são propostos, implementados e submetidos para competições a fim de medir e comparar desempenho tanto de memória quanto de tempo, frente a outros resolvedores e *benchmarks* variados. Como as heurísticas, implementações e parâmetros diferem muito entre resolvedores, é difícil de provar as razões que influenciaram um resolvedor a obter melhores resultados em comparação aos outros. Para prover melhores *insights* sobre as heurísticas, esta dissertação propõe

uma nova abordagem na construção de resolvedores de SAT, um resolvedor de SAT modular, que permite ao usuário a geração de tantos resolvedores quanto a combinação do módulos implementados.

A construção modular de resolvedores SAT não é nova, já que ela é encontrada no Minisat, através da herança da classe **Solver**. Mesmo assim, o tipo de modularidade proposta nesta dissertação melhora a organização, facilidade de modificação e simplicidade, já que as estruturas de dados específicas de cada heurística estão modularizadas em descrições XML, uma para cada heurística. Uma modularização similar da realizada por essa dissertação, é proposta por SATenstein [KhudaBukhsh et al., 2009]. Partindo da observação que desenvolver algoritmos para SAT é difícil e consome muito tempo, os autores propuseram um *framework* generalizado e altamente parametrizado, que inclui componentes de, ou baseados em, outros resolvedores. Os parâmetros controlam tanto a instanciação quanto o comportamento dos componentes. Um grande passo foi tomado pelos autores. Eles usam um algoritmo caixa-preta automatizado para configurar os parâmetros (veja [Hutter et al., 2007]) e assim encontrar instanciações do SATenstein de alta performance, de acordo com as instâncias de SAT fornecidas. As observações e objetivos do SATenstein são os mesmos que os do resolvedor modular proposto neste trabalho. As contribuições também são parecidas, já que ambas separam o desenho e a implementação do espaço de algoritmos, da busca, nesse espaço de algoritmos, daqueles que fornecem alta performance. Entretanto o SATenstein, tem uma contribuição maior, selecionando as instanciações e o comportamento automaticamente. Entretanto, não é claro como os módulos foram implementados. Na verdade, o SATenstein é baseado no SPEAR e PARAMILS [Hutter et al., 2007], mas aplicado à busca local estocástica. Diferentemente do SATenstein, a abordagem dessa dissertação prevê a combinação dinâmica dos módulos. Isso poderia ser implementado no SATenstein como um outro conjunto de parâmetros.

Outro tipo de modularidade de resolvedores é mostrada no SATzilla [Xu et al., 2008]. SATzilla é um resolvedor por instância com *portfolio*. Ele necessita de um conjunto predefinido de resolvedores e do tempo de execução relacionado a cada resolvedor para cada instância SAT que se deseja construir um modelo. O modelo gerado escolhe, dada características da entrada, qual resolvedor deve ser escolhido. O resolvedor proposto nessa dissertação pode ser integrado ao SATzilla, já que os resolvedores gerados pela abordagem proposta podem ser adicionados ao *portfolio*.

A possibilidade de utilizar a mesma heurística de dois resolvedores distintos em tempos diferentes de execução, é uma característica interessante da abordagem proposta, já que cria uma decisão dinâmica. Isso pode ser visualizado na Figura 4.1. No resolvedor *Solver* no tempo 0 tem-se as heurísticas do BerkMin, menos *DBManag* que

foi adaptada do Minisat. No tempo X, o método *DBManag* é trocado por um baseado no BerkMin, e o método *Decide* é trocado pelo método baseado no Minisat. O tempo de troca é definido por algum critério. Na implementação, a troca sempre ocorre após um reinício, já que o efeito de um reinício é o recomeço da busca. Isto evita possíveis inconsistências nas estruturas de dados, uma vez que é equivalente a aplicar um novo resolvedor ao conjunto prévio de cláusulas (aquelas que foram aprendidas e as originais do problema).

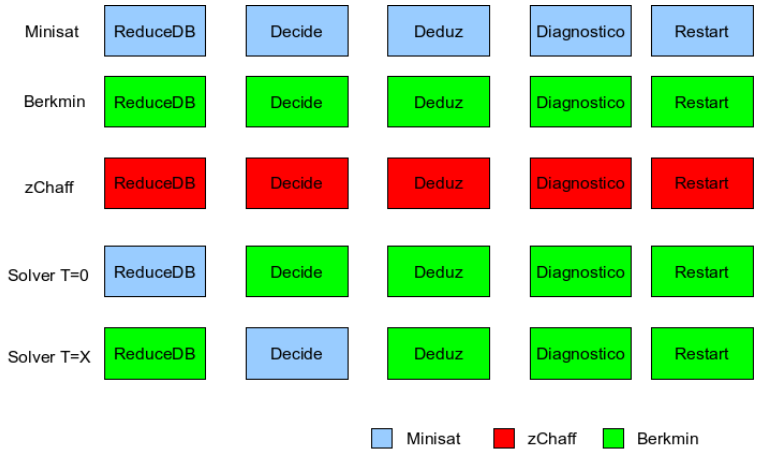


Figura 4.1. Utilização de Heurísticas Diferentes em Tempos Diferentes pelo Mesmo Resolvedor.

Para verificar efetividade da abordagem modular, foi escolhido um problema de SAT importante e complexo: o problema de Equivalência de Circuitos Combinacionais. Apesar de vários avanços feitos nas técnicas de resolvedores de SAT nas últimas décadas, algumas classes de circuitos, como os aritméticos continuam a ser um desafio. Por exemplo, provar a equivalência entre multiplicadores ou divisores usualmente gasta de duas a três ordens de magnitude mais tempo em comparação com outros circuitos combinacionais, como somadores, mesmo com o mesmo número de portas lógicas. Andrade et al[Andrade et al., 2008b] apresenta uma comparação entre os resolvedores de SAT estado-da-arte utilizando um *benchmark* com diversas instâncias CEC, derivadas de seu gerador de circuitos. Como demonstrado no artigo, multiplicadores e divisores continuam a desafiar o desempenho dos resolvedores de SAT. Outra conclusão feita no artigo é que o BerkMin era o resolvedor de SAT mais propício para o *benchmark*. Além disso, a prova de equivalência funcional entre multiplicadores de diferentes arquiteturas, cujas estruturas internas são pouco similares (por exemplo Dadda Tree x Wallace Tree, Array x Reduced Tree), tem uma complexidade maior que a verificação de multiplicadores com estrutura interna semelhante. Apesar de o BerkMin apresentar os melhores resultados no artigo referido, é bem difícil de demonstrar de forma teórica a

razão que leva-o a obter resultados tão bons, já que os resolvedores são implementados com diversas heurísticas e estruturas de dados diferentes. Muitas vezes essa diferença está em pequenos detalhes. Finalmente, o BerkMin é de código fechado, e muitos de seus detalhes de implementação podem não ter sido publicados.

Para prover uma oportunidade clara de novos avanços nas técnicas de resolvedores SAT, esta dissertação propõe, implementa e torna disponível, em código aberto, um novo resolvedor de SAT modular. A comunidade pode, agora, de forma mais livre modificar e testar heurísticas. Partindo dessa contribuição, esta dissertação mostra que apenas selecionando diferentes heurísticas, de modo modular, é possível atingir melhores tempos de execução que os obtidos por importantes e eficientes resolvedores, como o BerkMin, para diversas instâncias.

4.1.1 Hipóteses Testadas

A abordagem modular claramente cria um *framework* que permite que os usuários escolham, implementem e combinem as heurísticas como desejarem. Para estender as contribuições deste trabalho, três hipóteses foram levantadas para melhorar o tempo de execução do problema CEC por resolvedores de SAT, descritas em Enumeração 4.1.

A primeira hipótese consiste em usar uma heurística mais agressiva de redução da base de cláusulas aprendidas. O método, usado pelo Minisat, é mais agressivo pois somente mantém as cláusulas binárias da primeira metade do arranjo de cláusulas aprendidas e, da segunda metade, mantém apenas as cláusulas binárias com atividade maior que um fator do valor de decaimento da atividade da variável e do número de cláusulas. Menores cláusulas podem melhorar o tempo de execução já que um número menor de *cache misses* é gerado.

A segunda e terceira hipóteses permitem buscas em diferentes espaços de solução, o que é também conseguido através de reinícios. Reinícios e a aprendizagem de cláusulas são apontados, por [Gomes et al., 2008], como os responsáveis pela eficiência dos resolvedores atuais. Essas hipóteses testam se os espaços buscados por elas têm também impacto sobre a eficiência do resolvedor. A mudança dinâmica das heurísticas de decisão foi proposta em [Giunchiglia & Tacchella, 2004].

Outras duas instâncias do resolvedor proposto, sem hipóteses geradoras, são também apresentadas. Uma possui todas as heurísticas do BerkMin e do artigo Circuitos Dissimilares, enquanto a outra usa as heurísticas do Berkmin e do artigo Circuitos Dissimilares, menos as heurísticas de escolha do próximo literal e a de redução da base de cláusulas, que são adaptadas do Minisat.

Enumeração 4.1 Hipóteses a serem testadas utilizando a abordagem modular.

1. Mudar o BerkMin e o *Dissimilar Circuits* para usar o método de redução da base de cláusulas do Minisat;
2. Usar o BerkMin e o *Dissimilar Circuits* como base e mudar em tempo de execução as heurísticas de decisão entre o Minisat e o BerkMin;
3. Usar o BerkMin e o *Dissimilar Circuits* como base e mudar em tempo de execução a redução da base de cláusulas entre o do Minisat e o do BerkMin.

4.2 Decisões de Implementação

A premissa inicial, em relação a implementação, seria utilizar apenas herança de classes, influenciada pela forma de modularização do Minisat. O resolvedor baseado no BerkMin, chamado de *BerkSolver*, seria filho da classe *Solver*[1]. Para cada possível combinação dos módulos, haveria uma classe derivada da Classe BerkSolver e de Solver, como demonstrado por Solver1 e Solver2, descrito na Figura 4.2. Caberia decidir na implementação das Classes Solver1/Solver2 quais métodos das classes pais deveriam ser chamados. Porém, haveria um grave problema. Um grande número de arquivos poderia ser gerado, já que cada combinação dos métodos é implementada como uma classe, além, é claro, da produção de código extra específico para possibilitar a integração entre os métodos dos diferentes resolvedores.

Uma segunda abordagem seria adotar apenas herança de *Solver* ← *BerkSolver* e a implementar as modificações dos módulos através de *flags* de compilação. Isso eliminaria a necessidade de replicação de código, necessária no primeiro caso. O usuário utilizaria as *flags* para indicar qual combinação de heurísticas ele quer usar. A própria compilação com os *flags* escolhidos gerava um resolvedor que representa a combinação desejada. Porém, o código final ficaria de difícil entendimento, uma vez que não se teria uma visão global do que cada *flag* implementa ou modifica no programa. Além disto, o esquema de macros de C não permite *flags* encadeadas.

A terceira abordagem seria utilizar Orientação Por Aspectos. Nesse esquema o código seria incluído em algum evento pré-definido, como na chamada do método. Porém o objetivo da orientação por aspectos é a implementação de componentes ortogonais ao código, como requisitos não funcionais ou *logging*, e não geração de código. A solução final foi desenvolver um pré-processador que substitui pontos determinados pelo programador por código C/C++.

[1]Para informações dos métodos principais do resolvedor *Solver* veja a Seção 3.7.

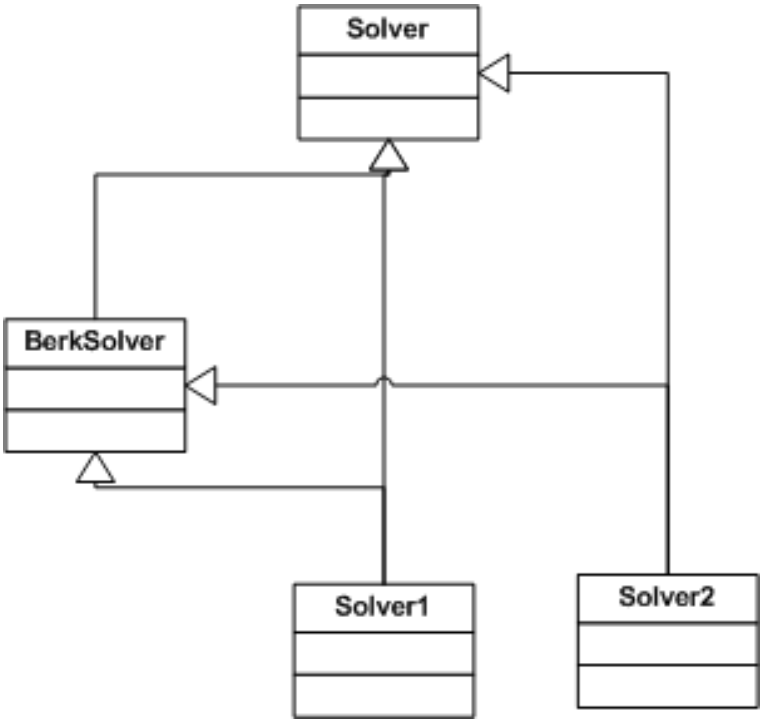


Figura 4.2. Herança Representando a Primeira Idéia de Implementação

4.2.1 Geração de Código no Pré-processamento

Para realização da geração de código, um programa Perl foi desenvolvido. O esquema geral de geração do código pode ser vista na Figura 4.3. Os componentes básicos da geração são dois: marcadores, também chamados de identificadores nos arquivo C++; e arquivos XML que definem o código que substituirá o marcador. Observando a Figura 4.3, começa-se marcando o Resolvedor Base (no caso o Minisat), gerando um conjunto de marcadores. Os outros resolvedores são divididos em heurísticas, que são reimplementadas em arquivos XML. Esses arquivos utilizam os marcadores pré-definidos como pontos de geração do código. O resolvedor marcado, as heurísticas e a descrição XML dos módulos que compõe o resolvedor a ser gerado são dados como entrada para o pré-processador Perl, que gera o resolvedor.

Por exemplo veja o Código Fonte 4.1. Esse fragmento possui dois identificadores: //<**!–USolver.h/declarations**–> e //<**!–USolver.h/implementations**–>. Um exemplo de arquivo XML que define o código a ser gerado é mostrado no Código Fonte 4.2. Todo arquivo de heurística começa com o *tag* **heuristic** e possui o atributo **id**, que contém o nome da heurística. Cada heurística possui um ou mais mapeamentos marcador → código, e uma ou mais sub-heurísticas. Cada mapeamento começa com o tag **macro**, e possui os atributos **id** indicando qual é o marcador; **type** que pode

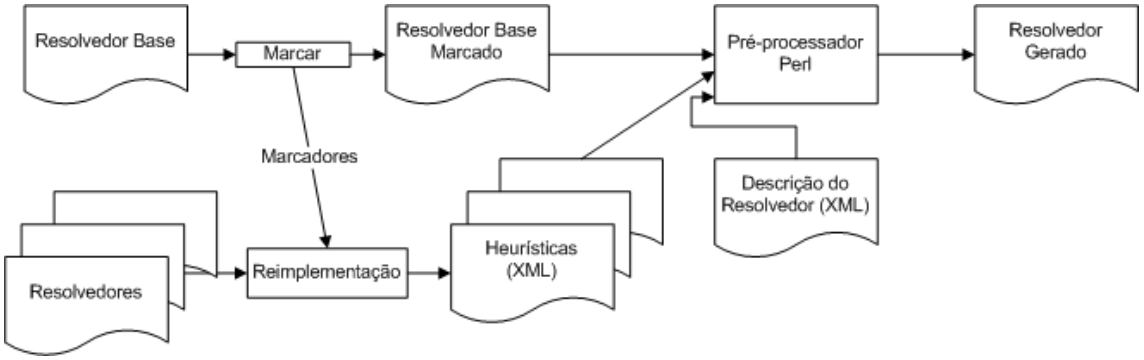


Figura 4.3. Implementação do Resolvedor Modular Utilizando Pré-processamento e Arquivos XML.

ser **append** para concatenação de código ou **overwrite** para substituição de código; e **priority** que indica qual a prioridade deste mapeamento. A prioridade é utilizada para resolver qual código será gerado no caso de um overwrite ou define a ordem no caso de haver apenas concatenação (ordenação crescente). O overwrite tem maior prioridade que qualquer concatenação, logo se for desejado concatenar código a um overwrite, o código do overwrite deve ser um marcador, que será substituído por outro mapeamento. Esse sim pode ser concatenado. Para descrição do código a ser gerado pelo mapeamento, dentro da tag **macro** há duas *tags* que podem ser usadas. A *tag* **code** compreende o código C++ enquanto a *tag* **file** possui o atributo **name** que indica o arquivo com o código C++. Cada sub-heurística é definida utilizando-se a *tag* **subheuristic**, que contém o atributo **id** que indica o arquivo que contém a heurística. Essa tag é demonstrada no Código Fonte 4.3.

Por definição, os resolvedores são conjuntos de heurísticas. Estes resolvedores também são definidos em arquivos XML, e tem a estrutura demonstrada no Código Fonte 4.4 que representa o resolvedor Minisat. A **tag** inicial é **solver**, que contém as **tags**: **output** com o atributo **folder** indicando a pasta de destino e a tag **heuristic** com o atributo **file** indicando o arquivo que contém a heurística.

```
1 class USolver: public Solver {
  //<!--USolver.h/declarations-->
  };
  //<!--USolver.h/implementations-->
```

Código Fonte 4.1. Fragmento do Arquivo USolver.h

```
  <heuristic id="dbManag">
  <macro id="USolver.h/declarations" type="append" priority="0">
  <code>
  protected:
    void minisat_reduceDB();
6   void      claDecayActivity ();
```

```
  void      claBumpActivity  (Clause& c);
</code>
</macro>
<macro id="USolver.h/implementations" type="append" priority="0">
<file name="minisat/dbmanag.implementation.h" />
</macro>
</heuristic>
```

Código Fonte 4.2. Fragmento do Arquivo minisat/dbmanag.xml

```
<heuristic id="decide_berkmin_minisat">
<subheuristic id="minisat/decide.xml"/>
<subheuristic id="berkmin/decide.xml"/>
</heuristic>
```

Código Fonte 4.3. Fragmento do Arquivo combination/decide.xml

```
<solver id="minisat">
<output folder="src-minisat"/>
<heuristic file="minisat/dbmanag.xml" />
<heuristic file="minisat/decide.xml"/>
<heuristic file="minisat/reducedb.xml"/>
<heuristic file="minisat/restart.xml"/>
<heuristic file="minisat/simplify.xml"/>
<heuristic file="minisat/var.xml"/>
</solver>
```

Código Fonte 4.4. Arquivo do resolvedor Minisat

4.3 Metodologia

Após decidida a abordagem de pré-processamento, uma metodologia para integração dos módulos foi desenvolvida. A idéia geral é identificar o que cada heurística representa, o que deve ser mudado no código, e quais estruturas de dados são necessárias. Por exemplo, as bases de cláusulas, tanto das originais quanto das aprendidas, não são replicadas, uma vez que é justamente o efeito que cada método tem sobre elas que se deseja manter, mesmo com a troca em tempo de execução. O contador de atividade de cada cláusula teve, entretanto, uma alteração. Nesse caso, incluiu-se um novo contador para cada cláusula chamado de berk_act para indicar a atividade da cláusula segundo o método do BerkMin. A metodologia é descrita no Figura 4.4.

Para exemplificar a metodologia, observe os Códigos Fontes 4.5 e 4.6. O Código Fonte 4.5 mostra vários aspectos da metodologia. A primeira macro *SolverTypes.h/Clause/declarations* cria o contador de atividades do BerkMin para cada cláusula. Esse contador precisa ser iniciado, macro *SolverTypes.h/Clause/Constructor/body*, e incrementado em alguns pontos do programa, macro *USolver.C/member/search/body4*

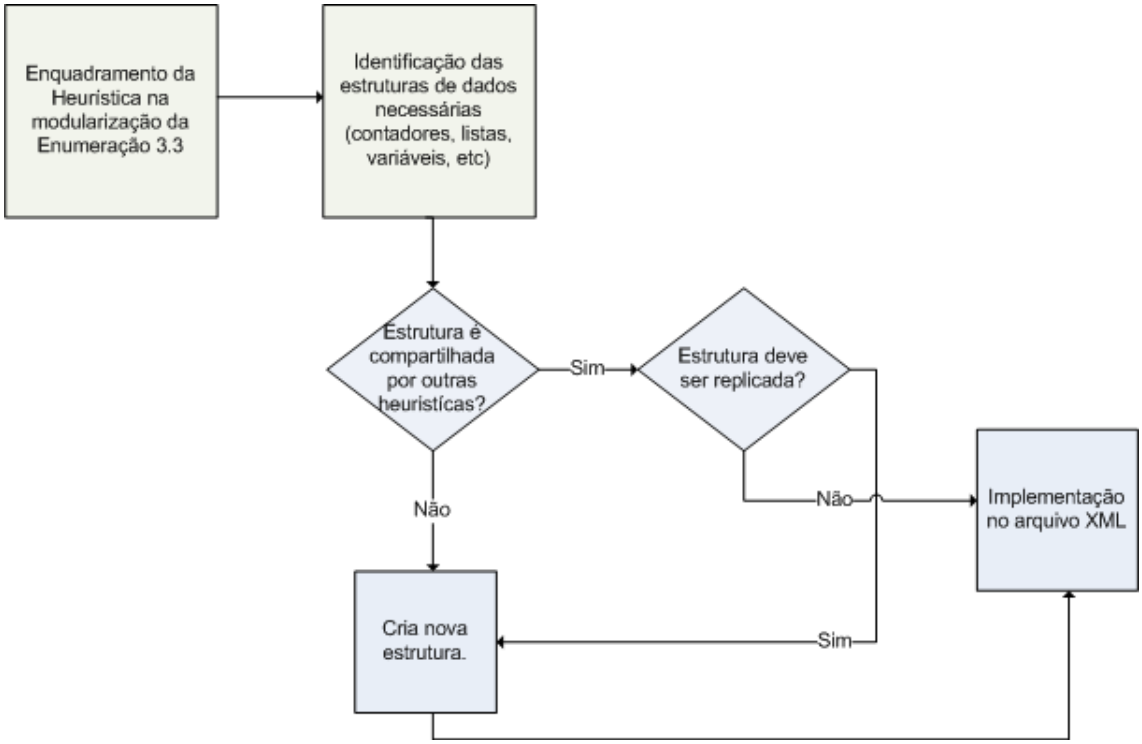


Figura 4.4. Metodologia de implementação dos módulos

e *USolver.C/member/analyze/c.learnt*. O contador é utilizado no método *USolver::berk_reduceDB()* para selecionar as cláusulas que devem ser removidas. O método é declarado na macro *USolver.h/declarations* e implementado na macro *USolver.C/implementations*. O Código Fonte 4.6 demonstra o que é feito quando a combinação dos métodos DBManag do BerkMin e do Minisat é ativada. O primeiro passo é incluir ambas heurísticas, sub-heurísticas *minisat/dbmanag.xml* e *berkmin/dbmanag.xml*. Para selecionar em tempo de execução qual heurística é utilizada uma nova variável é criada na macro *USolver.h/declarations* e iniciada na macro *USolver.C/member/Constructor/body*. O novo corpo do método reduceDB é determinado pela variável criada e é demonstrado no código da macro *USolver.C/member/reduceDB/body*. A troca periódica do valor da variável é feita na macro *USolver.C/member/solve/body2.1*, e ocorre a cada 50 reinícios.

```
1 <heuristic id="berkmin/dbManag">
  <macro id="SolverTypes.h/Clause/declarations" type="append" priority="0">
  <code>
  public:
    uint64_t berk_act;
    uint64_t& berk_activity(){return berk_act;}
  </code>
  </macro>
  <macro id="SolverTypes.h/Clause/Constructor/body" type="append" priority="0">
  <code>
```

```
11 berk_act = 0;
   </code>
   </macro>
   <macro id="USolver.C/member/search/body4" type="append" priority="0">
   <code>
                  berk_claBumpActivity(*c);
   </code>
   </macro>
   <macro id="USolver.C/member/analyze/c.learnt" type="append" priority="0">
   <code>
21       berk_claBumpActivity(c);
   </code>
   </macro>
   <macro id="USolver.h/declarations" type="append" priority="0">
   <code>
   protected:
     void    claBumpThreshold ();
     void      berk_claBumpActivity  (Clause& c);
     void berk_reduceDB();
   private:
31   double old_clause_act_threshold;
     double young_clause_act_threshold;
     double threshold_inc;
     int berk_cla_inc;
     //Threshold size of clauses values
     int oldclausethsize;
     int youngclausethsize;
   </code>
   </macro>
   <macro id="USolver.h/implementations" type="append" priority="0">
41 <code>
   inline void USolver::claBumpThreshold() {
     static const double dMax = std::numeric_limits<double>::max();
     if(old_clause_act_threshold != dMax)
       old_clause_act_threshold += threshold_inc;
   }

   inline void USolver::berk_claBumpActivity  (Clause& c){             // Increase a
       clause with the current 'bump' value.
     c.berk_activity()+=berk_cla_inc;
   }
51 </code>
   </macro>
   <macro id="USolver.C/implementations" type="append" priority="0">
   <file name="berkmin/dbmanag.implementation.C" />
   </macro>

   </heuristic>
```

Código Fonte 4.5. Fragmento de DBManag BerkMin

```
<heuristic id="dbmanag_berkmin_minisat">
<subheuristic id="minisat/dbmanag.xml"/>
```

```
<subheuristic id="berkmin/dbmanag.xml"/>
<macro id="USolver.h/declarations" type="append" priority="0">
<code>
  bool berkmin_db_manag;
</code>
</macro>
<macro id="USolver.C/member/Constructor/body" type="append" priority="0">
<code>
  this->berkmin_db_manag=true;
</code>
</macro>
<macro id="USolver.C/member/reduceDB/body" type="overwrite" priority="1">
<code>
  if(this->berkmin_db_manag)
    this->berk_reduceDB();
  else
    this->minisat_reduceDB();
</code>
</macro>
<macro id="USolver.C/member/solve/body2.1" type="append" priority="0">
<code>
  if(starts>0 && starts % 50 == 0){
    if(!this->berkmin_db_manag){
      //<!--USolver.C/member/solve/body2/sorttime-->
    this->old_top_clause_idx = -1;
    }else{
    ;
    }
  this->berkmin_db_manag = !this->berkmin_db_manag;
  }
</code>
</macro>
</heuristic>
```

Código Fonte 4.6. Fragmento de DBManag da combinação das macros Minisat BerkMin

4.4 Principais Estruturas de Dados

As estruturas de dados foram herdadas do Minisat. Os assinalamentos de variáveis são guardados em uma pilha, semelhante ao descrito na Seção 3.1. A cada assinalamento de variável de decisão o nível da busca é incrementado. A cada assinalamento das variáveis de decisão e das variáveis deduzidas pelo BCP, o número representando o literal é empilhado na pilha *trail*. A cada variável de decisão, um apontador para seu valor é empilhado na pilha $trail\_lim$, que funciona como um marcador.

A variável é representada pelo número inteiro lido do arquivo DIMACS. O literal da variável x é representado na classe Lit e seu valor é calculado como $2*x$ para o literal positivo e $2*x+1$ para o literal negativo. Para obtenção da variável correspondente ao literal x ou $\overline{x}$ basta fazer o deslocamento para a direita de 1 bit, pois $2*x >> 1 ==$

$2 * x + 1 >> 1 == x$. Isto torna fácil a indexação de arranjos, tanto por variáveis quanto por literais.

A base de cláusulas é dividida em duas, uma correspondendo à base de cláusulas originais e outra da base de cláusulas aprendidas pelo conflito. A diferença é que a segunda base de cláusulas pode ter cláusulas eliminadas durante a busca, já que a cláusula aprendida é sempre uma consequência lógica da base de cláusulas original[2]. A base de cláusulas aprendidas é implementada como um fila, tal que o último elemento é a cláusula aprendida mais recentemente.

Os contadores de atividade, das cláusulas e das variáveis, são implementados como arranjos de *double*, e são indexados ou pela variável ou pelo literal. Os contadores de atividades são utilizados tanto pelo BerkMin como pelo Minisat, tanto para as cláusulas quanto para as variáveis.

A estrutura de dados "Two-Watched Literals" é implementada como um arranjo de listas de cláusulas, indexado por literais. Um arranjo de inteiros chamado *reason*, indexado pelas variáveis, representa as cláusulas que levaram ao assinalamento da variável através do BCP. Esse arranjo é utilizado para construção do grafo de implicações.

4.5 Implementação das Heurísticas

Nas subseções a seguir adotou-se a nomenclatura das heurísticas baseadas nas Seção 3.3, e apresentada na Enumeração 3.5. Os arquivos que implementam as heurísticas tem o nome da heurística implementada. A relação completa das pastas e dos arquivos XML estão descritas no Código Fonte 4.7. A pasta corresponde ao resolvedor, e os arquivos XML às heurísticas. Por exemplo, as heurísticas do artigo *Dissimilar Circuits* estão na pasta *dcircuit*, e são *restart* (Reinício) e *var* (modificação no Decide do BerkMin).

```
.:
berkmin
combination
dcircuit
minisat
others

./berkmin:
dbmanag.implementation.C
dbmanag.xml
decide.implementation.C
decide.implementation.h
decide.xml
diagnose.xml
newparms.xml
```

[2]Veja Sub-seção 3.2.1.

```
parms.xml
reducedbcond.xml
restart.xml
simplify.xml
var.xml

./combination:
combination.xml
dbmanag.xml
decide.xml

./dcircuit:
restart.xml
var.xml

./minisat:
dbmanag.implementation.C
dbmanag.implementation.h
dbmanag.xml
decide.implementation.C
decide.xml
reducedbcond.xml
restart.xml
simplify.xml
var.xml

./others:
sort.xml
```

Código Fonte 4.7. Distribuição dos Arquivos XML

4.5.1 Decide

Esta heurística retorna a variável de decisão e seu valor. As heurísticas utilizadas são tanto do BerkMin, quanto do Dissimilar Circuits e do Minisat. As heurísticas implementadas no método do BerkMin são a Sensibilidade da Escolha da Variável, sub-seção 3.5.1.1, Mobilidade de Escolha da Variável, sub-seção 3.5.1.2 e Simetrização da Base, sub-seção 3.5.1.2. As do Minisat são apresentadas na seção 3.7 e as do Dissimilar Circuits na seção 3.6, e representa a escolha da variável na cláusula do topo mesmo quando a cláusula foi satisfeita.

Conforme apontado na seção anterior, algumas estruturas de dados foram replicadas. O vetor de atividade das variáveis é um deles. Há por exemplo um vetor de atividade das variáveis para o BerkMin e um vetor de atividade para o Minisat. As operações sobre o vetor, seguem o padrão: sufixo *berk_* para as operações sobre o vetor do BerkMin e o sufixo *minisat_* para os operações sobre o vetor do minisat (exceto as operações que modificam os contadores que não possuem sufixo). Essa replicação ocorre devido à diferente forma de incremento e de divisão desses contadores. O Mini-

sat incrementa a atividade da variável uma única vez, enquanto o BerkMin incrementa para todas as cláusulas utilizadas na derivação do 1-UIP[3].

Como a atividade da variável é vinculada com o procedimento de escolha, tanto a declaração do vetor quanto das operações sobre ele ficam condicionadas à utilização do arquivo *minisat/decide.xml* ou do arquivo *berkmin/decide.xml*.

O procedimento *pickBranchLit* aceita a utilização concomitante dos dois métodos, arquivo *combination/decide.xml*. Quando isto ocorre, o método a ser chamado é decidido em tempo de execução, através de uma variável que indica qual método é aplicado. Nessa implementação a troca ocorre a cada 50 reinícios.

O procedimento do BerkMin (*berk_ pickBranchLit*) utiliza a simetrização da base, mostrada na Seção 3.5.1.3.

Em ambos os procedimentos **Decide**, tanto do BerkMin quanto do Minisat, a próxima variável a ser escolhida é retirada de um *heap*. O heap do Minisat é mantido durante toda a execução, enquanto o do BerkMin é construído a cada mudança da base de cláusulas aprendidas, ou segundo uma condição controlada pelo arquivo *dcircuit/var.xml*. Se ele não é utilizado, o heap é reconstruído com as variáveis da cláusula, ainda não satisfeita, do topo da pilha. Se o arquivo é utilizado o heap é reconstruído com as variáveis da cláusula do topo da pilha que não possui todas as variáveis assinaladas.

Uma característica interessante apresentada na seção 3.7 é que o Minisat utiliza um método baseado no VSIDS (Seção 3.4.2). Nessa modificação, o contador de atividade é da variável, cujo valor é determinado por meio de um de quatro métodos. O primeiro é sempre tentar o literal positivo, o segundo é sempre tentar o literal negativo, o terceiro é o valor atribuído pelo usuário e o quarto é um método randômico.

4.5.2 Deduz

Esta heurística é basicamente a mesma para os dois resolvedores e utiliza a estrutura "Two-Watched Literals" apresentada na Seção 3.4.1. O principal método utilizado é o BCP e determina os valores de outras variáveis a partir das cláusulas que se tornaram unitárias devido as decisões anteriores.

4.5.3 ReduceDB (DBManag + ReduceDBCond)

Esta heurística representa a redução da base de dados aprendidas, bem como da condição que aciona o mecanismo. O método em si é chamado de *DBManag*, enquanto

[3]Mais sobre o incremento da atividade da variável na Seção 4.5.4.

a condição da chamada do método é *ReduceDBCond*. A implementação do BerkMin utiliza a Sub-Seção 3.5.1.3, arquivo *berkmin/dbmanag.xml*, a do Minisat utiliza a Seção 3.7, arquivo *minisat/dbmanag.xml*. As condições de chamada são implementadas pelos arquivos *berkmin/reducedbcond.xml* e *minisat/reducedbcond.xml*. No procedimento relativo ao Minisat, teve-se que controlar a chamada de ordenação da base de cláusulas. A ordenação pela atividade da cláusula ocorre se o *minisat/decide.xml* é ativado e o arquivo *berkmin/decide.xml* não é ativado. A macro responsável pela ativação da ordenação é *USolver.C/minisat_reduceDB/sort*. Se ambos estão ativos, a ordenação é decidida em tempo de execução no arquivo *others/sort.xml*. Se não há a ordenação quando o *berkmin/decide.xml* é ativado, o procedimento de redução da base do minisat passa a retirar todas as cláusulas de tamanho maior que dois da primeira metade da base, isto é, das cláusulas mais antigas, e a retirar todas as cláusulas de tamanho maior que dois que não tenham no mínimo um limite de atividade da outra metade, isto é das cláusulas mais novas. Isto é muito mais agressivo que o método implementado na operação *berk_reduceDB*.

O procedimento do BerkMin utiliza o contador *berk_act* contido em cada cláusula e implementa a lógica apresentada na Subseção 3.5.1.4.

As condições que definem a chamada ReduceDB são controladas pelos arquivos *berkmin/reducedbcond.xml* e *minisat/reducedbcond.xml*. A condição do BerkMin é a cada reinício, enquanto a do Minisat é quando o número de literais aprendidos chega a um limite, que é multiplicado a cada reinício.

4.5.4 Diagnostico

Essa heurística corresponde à derivação do grafo de implicações (Seção sec:zChaff), e por consequência do 1-UIP e do nível de backtracking. A atualização dos contadores de atividade das cláusulas e das variáveis é feita durante o procedimento *analyze*. Há duas formas de atualização dos contadores de variáveis. A do BerkMin é feita atualizando-se o contador para cada variável de cada cláusula utilizada na determinação do conflito (Seção 3.5.1.1). Já a do Minisat só conta a primeira vez que a variável aparece durante a derivação do 1-UIP. No Chaff é feita apenas para os contadores da cláusula de conflito gerada. O contador de cláusulas é atualizado a cada conflito em que a cláusula participa. O controle de atualização dos contadores é feito com os arquivos *minisat/var.xml* e *berkmin/var.xml*. O efeito do uso conjunto é a atualização dos contadores das duas formas.

O contador de atividade de cláusula é replicado. Os contadores de cláusulas do Minisat são divididos a cada conflito, enquanto os contadores do BerkMin (berk_act)

nunca são divididos. Cada vez que a cláusula aprendida é utilizada num conflito, seu contador é incrementado por ambos resolvedores.

Novamente, o Minisat tenta simplificar a cláusula aprendida, retirando literais redundantes (Seção 3.7). Isto foi mantido também nesta implementação do BerkMin (apesar de não ser assim no BerkMin original), pois a minimização foi feita para retirar redundâncias, produzindo cláusulas menores. Logo, devido ao menor número de *cache misses*, isso geraria resolvedores mais rápidos.

4.5.5 Reinício

O reinício (restart) é uma condição que reinicia a busca pela solução. O reinício retorna a busca para o nível zero e as cláusulas e contadores são mantidos. Logo a busca percorre caminhos diferentes do inicial. Para esta heurística, há os arquivos *minisat/-restart.xml*, *berkmin/restart.xml* e *dcircuit/restart.xml*. O arquivo *dcircuit/restart.xml* só pode aparecer combinado com o arquivo *berkmin/restart.xml* e é a única combinação permitida. O comportamento quando os três arquivos são ativadas não é bem definido.

A condição do BerkMin é a cada 500 conflitos e a do Minisat é quando o número de conflitos passa de um limite. Esse limite é multiplicado a cada reinício. O arquivo *dcircuit/restart.xml* implementa o método de reinício do Dissimilar Circuits. A condição é que, se é o segundo conflito e o nível de decisão é maior que 15 então faça um reinício fraco (Seção 3.6.2). Se a condição é igual a do BerkMin então faça um reinício forte.

4.5.6 Parâmetros

Experimentalmente verificou-se um ganho de desempenho ao utilizar o parâmetro de decaimento das variáveis com o valor $\frac{1}{0.995}$. O arquivo que ativa este valor é a *berkmin/newparms.xml*. Os parâmetros do BerkMin são ativados com o arquivo *berkmin/-parms.xml*.

4.5.7 Extras

Os arquivos *berkmin/simplify.xml* e *minisat/simplify.xml* controlam quando a simplificação da base de dados é chamada. No BerkMin é chamada a cada reinício e no Minisat quando um limite de literais das cláusulas aprendidas é obtido. A simplificação é feita sempre no nível zero de decisão e retira da base de cláusulas todas as cláusulas aprendidas que são satisfeitas pelas implicações de tamanho um obtidas durante os conflitos.

Capítulo 5

Resultados

Os testes foram feitos com multiplicadores, divisores e somadores obtidos através do aplicativo BenCGen, explicado em [Andrade, 2008]. Conforme descrito na Seção 2.4, esse benchmark foi utilizado ao invés do ISCAS 85, pois esse último não representa um desafio aos resolvedores estado-da-arte atuais e não tem a gama de circuitos, tanto em tamanho quanto variedade, disponíveis no BenCGen. As abreviações para cada tipo de circuito são definidas na Tabela 5.1.

Abreviatura	**Tipo**	**Método**
Cla	Multiplicador	Carry LookAhead Multiplier
Wall	Multiplicador	Wallace Tree Multiplier
Dadda	Multiplicador	Dadda Tree Multiplier
Array	Multiplicador	Array Multiplier
Reduced	Multiplicador	Reduced Tree Multiplier
Nrdivider	Divisor	Non Restoring Divider
Rdivider	Divisor	Restoring Divider
Rca	Somador	Ripple Carry Adder
Csad	Somador	Carry Save Adder
Clad	Somador	Carry LookAhead Adder

Tabela 5.1. Tabela com os Tipos de Circuito.

Para relembrar as hipóteses, a Enumeração 4.1 é replicada na Enumeração 5.1. Para se testar a primeira hipótese, o primeiro resolvedor, chamado de BerkMin-D, utiliza a implementação dos módulos do BerkMin e do Dissimilar Circuits. Porém o módulo de redução da base de cláusulas aprendidas ($DBManag$) é o implementado no Minisat. Os resolvedores BerkMin-DeD, BerkMin-De2D, BerkMin* e BerkMin-D2 são modificações do BerkMin-D. As modificações são respectivamente: usar a heurística de decisão do Minisat; modificar em tempo de execução a heurística de decisão ($Decide$);

Enumeração 5.1 Hipóteses a serem testadas utilizando a abordagem modular.

1. Mudar o BerkMin e o *Dissimilar Circuits* para usar o método de redução da base de cláusulas do Minisat;
2. Usar o BerkMin e o *Dissimilar Circuits* como base e mudar em tempo de execução as heurísticas de decisão entre o Minisat e o BerkMin;
3. Usar o BerkMin e o *Dissimilar Circuits* como base e mudar em tempo de execução a redução da base de cláusulas entre o do Minisat e o do BerkMin.

usar a redução das cláusulas aprendidas do BerkMin; e, modificar em tempo de execução, as heurísticas de redução da base de cláusulas aprendidas. O BerkMin-De2D testa a segunda hipótese e o BerkMin-D2 testa a terceira hipótese. O BerkMin* representa a implementação do resolvedor BerkMin estratégia 1. O BerkMin-DeD foi implementado apenas como um teste, sem nenhuma hipótese geradora.

Seguem as heurísticas utilizadas em cada resolvedor proposto:

- Resolvedor1: BerkMin-D (DBManag do Minisat)
  minisat/dbmanag.xml, berkmin/decide.xml, berkmin/reducedbcond.xml, berkmin/restart.xml, dcircuit/restart.xml, berkmin/simplify.xml, berkmin/var.xml, dcircuit/var.xml, berkmin/parms.xml, berkmin/newparms.xml

- Resolvedor2: BerkMin-DeD (Decide, DBManag do Minisat)
  minisat/dbmanag.xml, minisat/decide.xml, berkmin/reducedbcond.xml, berkmin/restart.xml, dcircuit/restart.xml, berkmin/simplify.xml, berkmin/var.xml, dcircuit/var.xml, berkmin/parms.xml, berkmin/newparms.xml

- Resolvedor3: BerkMin-De2D (Decide troca em tempo de compilação e DBManag do Minisat)
  minisat/dbmanag.xml, combination/decide.xml, berkmin/reducedbcond.xml, berkmin/restart.xml, dcircuit/restart.xml, berkmin/simplify.xml, berkmin/-var.xml, dcircuit/var.xml, berkmin/parms.xml, berkmin/newparms.xml

- Resolvedor4: BerkMin* (Somente módulos do BerkMin e Dissimilar circuits)
  berkmin/dbmanag.xml, berkmin/decide.xml, berkmin/reducedbcond.xml, berkmin/restart.xml, dcircuit/restart.xml, berkmin/simplify.xml, berkmin/var.xml, dcircuit/var.xml, berkmin/parms.xml, berkmin/newparms.xml

- Resolvedor5: BerkMin-D2 (troca em tempo de compilação DBManag)
  combination/dbmanag.xml, berkmin/decide.xml, berkmin/reducedbcond.xml,

berkmin/restart.xml, dcircuit/restart.xml, berkmin/simplify.xml, berkmin/-var.xml, dcircuit/var.xml, berkmin/parms.xml, berkmin/newparms.xml

O tempo é dado em segundos. As entradas com "-" indicam que não houve tempo hábil para terminar o teste. O tempo máximo de execução foi de dez mil segundos, as entradas com "*****" tiveram falha de segmentação. Quando ocorreu um *timeout*, os testes subseqüentes para o mesmo tipo de circuito foram abortados. Os resultados foram obtidos em um Athlon IV 3,0 GHz com 2 GB de memória rodando Fedora Core Linux OS 10.0. Apesar dos tempos de verificação poderem diminuir com o aumento do tamanho do circuito, foi determinado o limite de 10.000 segundos para os testes.

Para efeitos comparativos foram utilizados os resolvedores RSAT [Knot, 2007], BerkMin561 estratégia 1, Minisat e Siege_v4. Os parâmetros passados a cada resolvedor foram:

- minisat -verbosity=0 <arquivo>
- berkmin561 <arquivo> s 1 t 200000
- siege_v4 <arquivo> 100
- rsat <arquivo> -s -t 200000

Os nomes dos resolvedores foram abreviados. B-D representa os resultados do resolvedor BerkMin-D, B-DeD os resultados de BerkMin-DeD, B-De2D de BerkMin-De2D, B* de BerkMin* e B-D2 de BerkMin-D2. "Tam" representa o número de bits das entradas dos circuitos e "# Cl" representa o número de cláusulas do circuito *miter* resultante. Todas as entradas utilizadas nas tabelas são insatisfazíveis (UNSAT), já que os circuitos são equivalentes. As duas últimas colunas representam o SpeedUp do resolvedor BerkMin-D em relação ao Minisat e o BerkMin. Na explicação das tabelas serão utilizados os sufixos da Tabela 5.1.

As Tabelas 5.3, 5.2, 5.5, 5.6 e 5.4 são os resultados para duas cópias dos mesmos multiplicadores. Em uma das cópias foi aplicada a ferramenta de otimização ABC [Mishchenko, 2009]. As Tabelas 5.8, 5.7 e 5.9 representam a verificação de equivalência de um circuito Multiplicador Array com outro Multiplicador de estrutura diferente, respectivamente Dadda Tree, Carry LookAhead e Wallace Tree. A Tabela 5.10 é o resultado para verificação de um Multiplicador Carry LookAhead com um Wallace Tree. As tabelas 5.11 e 5.12 são os resultados da verificação de equivalência de cópias de divisores. Já as tabelas 5.13, 5.14 e 5.15 são os resultados da verificação de equivalência de cópias de somadores. Novamente em ambos os casos, somadores e divisores, como o mesmo circuito foi utilizado, a ferramenta ABC foi aplicada a uma das cópias.

Para alguns multiplicadores com estruturas internas semelhantes (Wall x Wall, Tabela 5.3; Dadda x Dadda, Tabela 5.2) o BerkMin-D perdeu para os outros resolvedores nos circuitos de tamanho 6x6 ou 7x7 bits, nos quais o resolvedor BerkMin se saiu melhor e o tempo de verificação é inferior a 5 segundos. Porém, com o aumento do tamanho do circuito, o BerkMin-D passou a apresentar bons resultados. Além disso ele passou a resolver mais instâncias do que os outros resolvedores, assumindo o limite de 10.000 segundos. Para essas tabelas, o BerkMin-D teve os melhores resultados em 46 das 56 instâncias, e o *SpeedUp* chegou até 3642,97% do BerkMin e 14088,40% do Minisat para Dadda e 1550,67% do BerkMin e 6318,05% do Minisat para Wall. O BerkMin-D2 foi melhor em 2 casos, no Wall 26x26 e 31x31. Para o circuito Reduced x Reduced, Tabela 5.5, a surpresa foi o resolvedor RSAT. Ele ultrapassou o limite de tempo em 12x12 bits. Porém fez o melhor tempo de 30 a 32 bits. O BerkMin-D novamente teve bons resultados, ganhando em 20 das 31 instâncias e chegando a 2217,90% de SpeedUp comparado ao BerkMin e 165933,90 % comparado ao Minisat. O BerkMin-D2 foi melhor em 22x22 e 25x25. Para os outros casos de multiplicadores com estruturas internas semelhantes (Array x Array Tabela 5.6, Cla x Cla Tabela 5.4) o BerkMin se saiu melhor quando o tamanho do circuito aumentava. Nesses multiplicadores, o BerkMin-D ganhou em 15 das 43 instâncias.

Pode-se observar na Tabela 5.2 que os resultados não são monotonicamente crescentes, já que há casos em que circuitos de tamanho maior gastam menos tempo para serem verificados. Essa observação indica que o melhor seria executar os resolvedores para todas as instâncias, mesmo quando o limite de tempo fosse alcançado. Para o RSAT, na Tabela 5.5, foi permitida a continuação dos testes mesmo com *timeout*. O tempo de verificação do RSAT variou bastante entre 11x11 e 14x14. Após o *timeout* que ocorre em 12x12, o RSAT foi o que obteve o melhor tempo de execução para o 32x32.

Resultados CEC para duas cópias de Multiplicadores Dadda Tree											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
2x2	260	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-100,00
3x3	518	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-50,02
4x4	956	0,01	0,01	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	83,33
5x5	1570	0,09	0,08	0,07	**0,04**	0,06	0,06	0,07	**0,04**	0,06	-33,32	28,34
6x6	2376	0,58	0,71	0,61	0,41	1,15	5,90	1,27	0,38	**0,37**	-64,28	-38,07
7x7	3390	8,20	**3,53**	6,70	3,89	4,69	170,10	6,93	3,57	3,79	-17,12	-24,88
8x8	4628	77,16	31,46	105,45	82,35	**9,09**	1525,65	10,47	56,10	63,47	805,88	246,12
9x9	6106	1387,23	203,47	1290,37	218,44	**17,00**	9838,35	17,27	556,73	576,67	1184,99	1096,92
10x10	7840	1541,15	1881,93	>10000	262,67	**30,22**	>10000	38,11	529,95	496,22	769,10	6126,73
11x11	9846	2834,09	4652,24	-	784,85	**42,45**	-	91,76	442,64	466,49	1748,90	10859,47
12x12	12140	>10000	3509,96	-	481,72	**48,21**	-	216,68	360,31	271,53	899,18	7180,31
13x13	14738	-	>10000	-	2848,30	**76,10**	-	340,74	279,64	247,10	3642,97	-
14x14	17656	-	-	-	1254,54	**98,77**	-	668,77	278,50	291,31	1170,11	-
15x15	20910	-	-	-	1543,26	**145,63**	-	1413,83	260,93	292,79	959,72	-
16x16	24516	-	-	-	4918,16	**128,95**	-	1257,80	379,92	421,59	3713,86	-
17x17	28490	-	-	-	5221,15	**175,48**	-	1936,88	410,94	599,63	2875,35	-
18x18	32848	-	-	-	2533,93	**348,49**	-	2431,00	623,36	556,40	627,12	-
19x19	37606	-	-	-	3399,27	**402,18**	-	3299,95	869,57	787,67	745,22	-
20x20	42780	-	-	-	6994,58	**421,85**	-	4849,32	984,71	957,50	1558,08	-
21x21	48386	-	-	-	6900,05	**853,40**	-	6184,72	1290,80	1281,18	708,54	-
22x22	54440	-	-	-	8356,76	**1072,16**	-	6063,27	1966,52	1434,36	679,43	-
23x23	60958	-	-	-	6570,74	**1088,50**	-	>10000	2138,23	2099,38	503,65	-
24x24	67956	-	-	-	6188,23	**2368,56**	-	-	3372,83	2398,17	161,26	-
25x25	75450	-	-	-	8587,18	**2441,56**	-	-	3861,71	3410,28	251,71	-
26x26	83456	-	-	-	7420,23	4043,48	-	-	5085,72	**3816,83**	83,51	-
27x27	91990	-	-	-	>10000	**2668,93**	-	-	5366,22	5172,39	-	-
28x28	101068	-	-	-	-	**3703,77**	-	-	6278,46	5362,70	-	-
29x29	110706	-	-	-	-	**5007,96**	-	-	7874,97	6916,72	-	-
30x30	120920	-	-	-	-	**5825,88**	-	-	>10000	7416,87	-	-
31x31	131726	-	-	-	-	9445,27	-	-	-	**9298,61**	-	-
32x32	143140	-	-	-	-	>10000	-	-	-	>10000	-	-

Tabela 5.2. Resultados CEC para duas cópias de Multiplicadores Dadda Tree

Resultados CEC para Duas Cópias de Multiplicadores Wallace Tree											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3x3	534	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-50,02
4x4	1060	0,01	0,01	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	83,33
5x5	1788	0,08	0,06	0,06	**0,04**	0,05	0,08	0,06	0,05	0,06	-24,52	13,21
6x6	3116	0,78	0,68	0,59	0,38	1,00	9,78	1,40	**0,32**	0,50	-61,96	-32,43
7x7	4392	7,07	3,46	6,57	3,37	**3,32**	207,14	4,34	3,40	4,49	1,52	4,25
8x8	5968	71,76	30,59	64,47	55,56	**6,48**	1872,90	8,69	53,54	49,44	757,27	372,03
9x9	7882	656,79	287,09	1487,46	220,15	**13,34**	>10000	20,14	482,49	682,68	1550,67	2052,60
10x10	9980	4742,42	1585,40	>10000	265,48	**24,70**	-	34,36	459,99	1158,87	974,72	6318,05
11x11	12422	>10000	>10000	-	320,36	**45,85**	-	109,77	692,03	418,23	598,73	-
12x12	15254	-	-	-	494,92	**45,30**	-	224,70	659,74	655,67	992,54	-
13x13	18390	-	-	-	960,39	**95,36**	-	315,44	662,51	1785,76	907,12	-
14x14	21808	-	-	-	1131,38	**198,66**	-	1544,03	1349,53	2281,18	469,50	-
15x15	27102	-	-	-	1601,91	**256,38**	-	2087,33	556,87	1552,34	524,81	-
16x16	30084	-	-	-	2213,73	**345,93**	-	2784,49	1032,62	2974,51	539,94	-
17x17	34732	-	-	-	2762,06	**427,32**	-	>10000	1277,02	3241,66	546,36	-
18x18	41576	-	-	-	3006,49	**598,57**	-	-	1833,40	4736,13	402,28	-
19x19	47272	-	-	-	3292,24	**912,64**	-	-	2644,37	9565,28	260,74	-
20x20	51342	-	-	-	5080,99	**1233,60**	-	-	3888,64	>10000	311,88	-
21x21	60062	-	-	-	5465,91	**1277,15**	-	-	4527,72	-	327,98	-
22x22	67218	-	-	-	6094,13	**3912,35**	-	-	4964,16	-	55,77	-
23x23	72188	-	-	-	7438,48	**2678,31**	-	-	6538,85	-	177,73	-
24x24	83200	-	-	-	6999,70	**2144,14**	-	-	8374,54	-	226,46	-
25x25	88810	-	-	-	>10000	**3518,17**	-	-	>10000	-	-	-
26x26	101126	-	-	-	-	**4311,21**	-	-	-	-	-	-
27x27	111040	-	-	-	-	**7892,15**	-	-	-	-	-	-
28x28	121400	-	-	-	-	**7234,22**	-	-	-	-	-	-
29x29	128386	-	-	-	-	>10000	-	-	-	-	-	-

Tabela 5.3. Resultados CEC para Duas Cópias de Multiplicadores Wallace Tree

Resultados CEC para Duas Cópias de Multiplicadores Carry LookAhead											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
2x2	118	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-100,00
3x3	326	0,01	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	0,00
4x4	686	0,02	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	42,90	14,29
5x5	1242	0,16	0,07	0,07	0,10	**0,05**	0,09	**0,05**	**0,05**	0,06	104,11	46,94
6x6	2046	0,98	0,44	0,44	0,77	1,59	4,72	2,24	**0,36**	0,38	-51,66	-72,19
7x7	3158	2,43	**2,17**	6,03	3,69	8,05	6816,20	9,90	3,28	4,99	-54,17	-73,10
8x8	4646	**18,32**	22,95	98,00	32,42	40,80	>10000	45,74	42,51	30,68	-20,54	-43,76
9x9	6586	**49,53**	91,34	1075,73	147,64	104,06	-	111,40	203,44	285,00	41,89	-12,22
10x10	9062	952,26	485,92	3416,99	386,06	**216,80**	-	666,55	868,39	1909,47	78,07	124,13
11x11	12166	2689,81	8804,55	4680,32	586,51	**372,31**	-	5029,80	2505,13	4012,24	57,53	2264,86
12x12	15998	>10000	>10000	>10000	**829,24**	838,30	-	>10000	9147,69	>10000	-1,08	-
13x13	20666	-	-	-	**1259,00**	1553,61	-	-	>10000	-	-18,96	-
14x14	26286	-	-	-	**2030,42**	2505,96	-	-	-	-	-18,98	-
15x15	32982	-	-	-	**2458,15**	4639,29	-	-	-	-	-47,02	-
16x16	40886	-	-	-	**3494,29**	8930,68	-	-	-	-	-60,87	-
17x17	50138	-	-	-	**5203,19**	>10000	-	-	-	-	-	-
18x18	60886	-	-	-	**6880,34**	-	-	-	-	-	-	-
19x19	73286	-	-	-	**9482,06**	-	-	-	-	-	-	-
20x20	87502	-	-	-	>10000	-	-	-	-	-	-	-
21x21	103706	-	-	-	-	-	-	-	-	-	-	-
22x22	122078	-	-	-	-	-	-	-	-	-	-	-
23x23	142806	-	-	-	-	-	-	-	-	-	-	-
24x24	166086	-	-	-	-	-	-	-	-	-	-	-
25x25	192122	-	-	-	-	-	-	-	-	-	-	-
26x26	221126	-	-	-	-	-	-	-	-	-	-	-

Tabela 5.4. Resultados CEC para Duas Cópias de Multiplicadores Carry LookAhead

Resultados CEC para Duas Cópias de Multiplicadores Reduced											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3x3	538	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-33,34
4x4	996	0,03	0,01	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	25,01
5x5	1630	0,16	0,07	0,06	**0,03**	0,06	0,06	0,07	0,04	0,05	-49,99	23,33
6x6	2436	1,39	0,49	0,63	0,33	0,67	26,17	0,73	**0,31**	0,40	-50,66	-27,05
7x7	3470	4,88	**2,77**	4,71	3,30	3,24	237,09	4,11	3,13	3,64	1,77	-14,71
8x8	4708	65,80	26,04	94,94	61,57	**4,71**	2328,76	7,29	48,10	61,43	1207,14	452,88
9x9	6186	824,76	386,20	2029,49	125,42	**13,38**	>10000	15,49	332,28	286,61	837,65	2787,28
10x10	7940	1919,47	1562,17	>10000	227,49	**14,65**	-	35,71	324,29	321,79	1452,85	10563,42
11x11	9946	1321,71	6505,13	-	244,38	**38,97**	-	73,83	256,80	158,62	527,14	16593,90
12x12	12240	10382,27	>10000	-	618,51	**41,29**	-	131,02	233,74	216,11	1398,01	-
13x13	14838	2165,25	-	-	740,78	**58,20**	-	257,17	343,78	253,21	1172,73	-
14x14	17776	1777,10	-	-	1181,76	**82,39**	-	311,38	192,04	347,67	1334,38	-
15x15	21030	3176,80	-	-	2065,70	**89,12**	-	1859,89	281,89	255,19	2217,90	-
16x16	24636	2070,83	-	-	2081,61	**304,85**	-	1144,45	310,37	390,20	582,83	-
17x17	28610	4424,13	-	-	3159,90	**205,49**	-	1761,85	422,99	533,46	1437,75	-
18x18	32968	3033,05	-	-	5837,93	**264,13**	-	3637,14	482,37	670,63	2110,21	-
19x19	37726	3630,20	-	-	4769,99	**306,62**	-	3109,64	591,31	722,29	1455,68	-
20x20	42920	4429,55	-	-	3473,10	**589,65**	-	2606,35	867,19	1197,20	489,01	-
21x21	48526	2485,11	-	-	7393,18	**847,39**	-	4905,38	1159,90	1286,18	772,46	-
22x22	54580	2958,35	-	-	9663,10	1355,38	-	>10000	1816,18	**1329,67**	612,94	-
23x23	61098	2119,25	-	-	9607,50	**1080,12**	-	-	2045,85	1916,74	789,49	-
24x24	68096	2838,08	-	-	-	**1512,69**	-	-	2875,01	2682,05	-	-
25x25	75590	5730,04	-	-	-	2997,02	-	-	3565,46	**2731,52**	-	-
26x26	83596	3803,74	-	-	-	**2033,08**	-	-	4799,52	4010,27	-	-
27x27	92130	4133,98	-	-	-	**3172,75**	-	-	6046,25	4244,73	-	-
28x28	101208	5539,58	-	-	-	**3769,33**	-	-	7826,23	5873,26	-	-
29x29	110866	4902,77	-	-	-	**4782,99**	-	-	8060,03	8434,01	-	-
30x30	121080	**6007,75**	-	-	-	6958,74	-	-	>10000	7547,60	-	-
31x31	131886	**7018,96**	-	-	-	8242,75	-	-	-	>10000	-	-
32x32	143300	**7007,77**	-	-	-	7223,00	-	-	-	-	-	-

Tabela 5.5. Resultados CEC para Duas Cópias de Multiplicadores Reduced

Resultados CEC para Duas Cópias de Multiplicadores Array											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
2x2	74	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-	-
3x3	230	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	0,00
4x4	466	0,01	0,01	0,01	**0,00**	0,01	0,01	0,01	**0,00**	0,01	-100,00	20,00
5x5	782	0,09	0,05	0,04	**0,02**	0,04	0,06	0,04	0,03	0,04	-44,44	47,22
6x6	1178	**0,20**	0,41	0,34	0,24	1,04	98,87	1,21	0,24	0,39	-76,94	-60,33
7x7	1654	1,36	**1,19**	3,52	3,10	5,68	1780,73	5,69	2,31	2,23	-45,41	-79,03
8x8	2210	7,02	**5,09**	34,89	14,31	9,27	4661,45	10,97	21,33	16,41	54,41	-45,06
9x9	2846	36,61	39,90	279,05	46,19	**13,64**	>10000	16,76	61,41	99,91	238,64	192,54
10x10	3562	125,69	231,46	1132,11	58,29	**30,99**	-	92,19	373,89	292,21	88,12	647,00
11x11	4358	353,87	996,44	3681,83	93,30	**47,76**	-	1151,11	584,73	1089,14	95,33	1986,13
12x12	5234	3494,78	>10000	5113,73	172,18	**80,68**	-	921,23	1015,49	1948,56	113,42	-
13x13	6190	>10000	-	>10000	**184,57**	213,32	-	>10000	2517,88	6696,43	-13,48	-
14x14	7226	-	-	-	383,00	**365,10**	-	-	4004,40	8915,30	4,90	-
15x15	8342	-	-	-	504,66	**386,54**	-	-	4739,63	>10000	30,56	-
16x16	9538	-	-	-	925,40	**719,51**	-	-	6049,37	-	28,61	-
17x17	10814	-	-	-	**1039,89**	1097,07	-	-	>10000	-	-5,21	-
18x18	12170	-	-	-	1803,50	**1617,31**	-	-	-	-	11,51	-
19x19	13606	-	-	-	2364,18	**2256,44**	-	-	-	-	4,78	-
20x20	15122	-	-	-	**3206,18**	5227,89	-	-	-	-	-38,67	-
21x21	16718	-	-	-	**3651,68**	4853,92	-	-	-	-	-24,77	-
22x22	18394	-	-	-	**5621,83**	6421,71	-	-	-	-	-12,46	-
23x23	20150	-	-	-	**6236,09**	9655,02	-	-	-	-	-35,41	-
24x24	21986	-	-	-	**8961,05**	>10000	-	-	-	-	-	-
25x25	23902	-	-	-	>10000	-	-	-	-	-	-	-

Tabela 5.6. Resultados CEC para Duas Cópias de Multiplicadores Array

Para os multiplicadores com estruturas internas diferentes (Tabelas 5.8, 5.7, 5.9, 5.10) os resultados foram bem diferentes dos anteriores. O BerkMin foi melhor para os circuitos de maior tamanho apenas no caso Array x Cla. Para o Array x Dadda, Array x Wall e Cla x Wall o Minisat se sobressaiu. Percebe-se nesses casos como o problema foi dificultado uma vez que apenas circuitos de pouco tamanho foram testados dentro do limite de tempo. Enquanto no caso Reduced x Reduced, resolve-se o problema até o tamanho 32x32 bits, nos circuitos em que o Minisat se sobressaiu não passou de 12x12 bits. Nos resultados de Array x Dadda, Array x Wall e Cla x Wall, percebe-se como a mudança proposta no BerkMin-D2 ajudou na redução no tempo em relação ao BerkMin-D. Os tempos obtidos pelo BerkMin* também foram melhores que o BerkMin-D.

Resultados CEC para um Multiplicador Array e um Carry LookAhead											**B-D SpeedUp(%)**	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
2x2	96	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-	-
3x3	278	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	0,00
4x4	576	0,01	0,01	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	-12,50
5x5	1012	0,09	0,06	0,07	**0,04**	0,05	0,08	0,05	**0,04**	**0,04**	-14,88	25,53
6x6	1612	0,37	**0,27**	0,70	0,31	0,82	132,64	1,31	0,35	0,37	-62,05	-66,46
7x7	2406	2,96	**1,98**	4,54	3,08	5,14	5571,34	12,55	2,72	2,74	-40,06	-61,43
8x8	3428	13,06	**6,79**	58,27	33,41	21,99	>10000	17,99	29,04	26,46	51,96	-69,13
9x9	4716	45,89	61,16	404,22	118,68	**44,16**	-	77,56	165,24	176,14	168,77	38,50
10x10	6312	448,42	457,22	3136,47	135,11	**76,17**	-	334,39	549,53	442,77	77,37	500,22
11x11	8262	987,74	3263,15	8099,01	229,32	**184,07**	-	2330,85	1431,44	2775,08	24,58	1672,73
12x12	10616	4222,52	>10000	>10000	349,05	**304,23**	-	9597,38	2923,94	9357,39	14,73	-
13x13	13428	>10000	-	-	619,86	**577,35**	-	>10000	4796,51	>10000	7,36	-
14x14	16756	-	-	-	**811,74**	1082,83	-	-	>10000	-	-25,04	-
15x15	20662	-	-	-	**1143,01**	1548,96	-	-	-	-	-26,21	-
16x16	25212	-	-	-	**1969,18**	2991,03	-	-	-	-	-34,16	-
17x17	30476	-	-	-	**2582,18**	4418,67	-	-	-	-	-	-
18x18	36528	-	-	-	**3762,86**	8180,27	-	-	-	-	-	-
19x19	43446	-	-	-	**4713,17**	>10000	-	-	-	-	-	-
20x20	51312	-	-	-	**6795,30**	-	-	-	-	-	-	-
21x21	60212	-	-	-	**8642,09**	-	-	-	-	-	-	-
22x22	70236	-	-	-	>10000	-	-	-	-	-	-	-

Tabela 5.7. Resultados CEC para um Multiplicador Array e um Carry LookAhead

Resultados CEC para um Multiplicador Array e um Dadda Tree											**B-D SpeedUp(%)**	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
2x2	167	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-	-
3x3	374	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	-50,02
4x4	711	0,01	0,01	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	0,00
5x5	1176	0,09	0,08	0,05	**0,03**	0,04	0,04	0,05	0,04	0,04	-24,99	107,50
6x6	1777	0,81	0,59	0,46	**0,30**	1,44	2608,54	2,90	0,31	0,37	-79,16	-58,75
7x7	2522	10,02	4,03	5,50	4,10	47,39	>10000	78,74	**3,55**	4,15	-91,35	-91,50
8x8	3419	126,38	**32,73**	80,99	120,71	3188,23	-	2639,03	80,83	74,78	-96,21	-98,97
9x9	4476	1627,48	**251,40**	1760,54	1418,48	>10000	-	>10000	1033,56	1122,70	-	-
10x10	5701	>10000	**1631,00**	>10000	>10000	-	-	-	8506,28	>10000	-	-
11x11	7102	-	>10000	-	-	-	-	-	>10000	-	-	-

Tabela 5.8. Resultados CEC para um Multiplicador Array e um Dadda Tree

Resultados CEC para um Multiplicador Array e um Wallace Tree											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3x3	167	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	200,20
4x4	374	0,01	0,02	0,01	**0,00**	0,01	0,01	0,01	0,01	0,01	-100,00	320,00
5x5	711	0,09	0,14	0,06	**0,03**	0,04	0,19	0,04	0,04	0,04	-24,99	252,51
6x6	1176	0,74	1,02	0,44	0,32	1,06	24,21	1,08	**0,31**	0,37	-69,81	-4,06
7x7	1777	9,50	**3,44**	5,13	3,58	45,65	>10000	81,05	3,95	5,06	-92,16	-92,47
8x8	2522	126,45	**33,89**	67,86	97,45	3393,13	-	4075,00	79,30	77,54	-97,13	-99,00
9x9	3419	1591,13	**222,71**	4635,66	1577,57	>10000	-	>10000	1072,25	1091,71	-	-
10x10	4476	>10000	**2184,22**	>10000	>10000	-	-	-	>10000	>10000	-	-
11x11	5701	-	**4194,28**	-	-	-	-	-	-	-	-	-
12x12	7102	-	>10000	-	-	-	-	-	-	-	-	-

Tabela 5.9. Resultados CEC para um Multiplicador Array e um Wallace Tree

Resultados CEC para um Multiplicador Carry LookAhead e um Wallace Tree											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3x3	430	**0,00**	**0,00**	0,01	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	-100,00	300,30
4x4	873	**0,01**	0,02	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	**0,01**	66,69	283,33
5x5	1515	0,12	0,17	0,06	0,08	**0,05**	0,07	**0,05**	**0,05**	**0,05**	77,81	286,67
6x6	2581	1,03	1,36	0,51	0,78	2,38	419,71	3,53	**0,36**	0,42	-67,15	-42,78
7x7	3775	10,43	5,62	5,36	**4,27**	68,51	>10000	83,43	4,76	5,69	-93,77	-91,80
8x8	5307	133,73	**48,52**	112,94	88,65	1708,47	-	2406,26	72,14	94,36	-94,81	-97,16
9x9	7234	1846,82	**385,65**	6036,97	1162,82	>10000	-	>10000	1229,12	1222,10	-	-
10x10	9521	>10000	**4023,06**	>10000	>10000	-	-	-	>10000	>10000	-	-
11x11	12294	-	>10000	-	-	-	-	-	-	-	-	-

Tabela 5.10. Resultados CEC para um Multiplicador Carry LookAhead e um Wallace Tree

Uma surpresa foi o caso do Array x Cla comparado ao Cla x Cla, já que o número de instâncias resolvidas pelo Array x Cla foi maior. Era de se esperar que ele tivesse menos instâncias resolvidas, uma vez que possui características internas menos semelhantes.

Para os divisores (Rdivider x Rdivider, Tabela 5.11 e Nrdivider x Nrdivider, Tabela 5.12), o Siege foi dominante para o Rdivider x Rdivider e o BerkMin para o Nrdivider x Nrdivider. O BerkMin* foi melhor em 4 das 25 instâncias do Nrdivider e BerkMin-D em 9 das 59 instâncias. Aponta-se, nesses divisores, o único caso em que o BerkMin-De2D se sobressaiu, Rdivider x Rdivider 12x12, porém mesmo assim, a diferença de tempo foi pequena comparada ao BerkMin.

Resultados CEC para Duas Cópias de Divisores Restoring											**B-D SpeedUp(%)**	
Divisor Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3	975	0,01	0,01	0,01	**0,00**	0,02	0,01	0,01	0,01	0,01	-100,00	-54,17
4	1680	0,09	0,08	**0,07**	**0,07**	0,47	0,08	0,26	0,09	0,10	-84,94	-82,15
5	2577	0,88	0,51	1,23	**0,35**	2,47	84,25	3,31	0,55	0,50	-85,81	-79,21
6	3666	**2,07**	3,13	5,64	2,31	4,24	80,00	3,72	4,06	2,43	-45,50	-26,11
7	4947	9,04	36,45	61,91	**6,33**	6,41	1373,46	7,31	17,18	16,22	-1,26	468,54
8	6420	116,31	306,09	129,03	16,63	**9,09**	>10000	14,09	52,36	73,51	82,88	3266,04
9	8085	437,69	1938,38	160,21	23,47	**21,39**	-	24,13	98,63	166,34	9,75	8964,33
10	9942	3034,56	>10000	225,67	66,78	**36,87**	-	77,19	167,53	268,21	81,10	-
11	11991	>10000	-	287,38	105,64	**81,60**	-	128,77	302,94	532,97	29,45	-
12	14232	-	-	372,23	166,67	212,14	-	**166,00**	767,81	902,54	-21,44	-
13	16665	-	-	410,34	**219,84**	342,39	-	250,60	1031,49	1911,27	-35,79	-
14	19290	-	-	461,39	**338,63**	876,94	-	1198,49	1709,71	4424,24	-61,38	-
15	22107	-	-	575,18	**441,38**	1492,93	-	811,07	2050,21	6051,82	-70,44	-
16	25116	-	-	**605,38**	735,04	3345,12	-	1553,56	3560,62	7670,48	-78,03	-
17	28317	-	-	**707,77**	889,32	4412,40	-	2829,98	5764,16	>10000	-79,84	-
18	31710	-	-	**755,20**	1079,88	4938,88	-	3731,32	9825,55	-	-78,14	-
19	35295	-	-	**830,89**	1578,49	5192,80	-	>10000	>10000	-	-69,60	-
20	39072	-	-	**906,25**	2281,96	9405,56	-	-	-	-	-75,74	-
21	43041	-	-	**1008,70**	2728,45	9249,43	-	-	-	-	-70,50	-
22	47202	-	-	**1101,45**	3202,05	>10000	-	-	-	-	-	-
23	51555	-	-	**1151,58**	4080,39	-	-	-	-	-	-	-
24	56100	-	-	**1254,74**	6080,52	-	-	-	-	-	-	-
25	60837	-	-	**1406,83**	6743,08	-	-	-	-	-	-	-
26	65766	-	-	**1539,17**	8208,84	-	-	-	-	-	-	-
27	70887	-	-	**1550,74**	>10000	-	-	-	-	-	-	-
28	76200	-	-	**1663,38**	-	-	-	-	-	-	-	-
29	81705	-	-	**1781,81**	-	-	-	-	-	-	-	-
30	87402	-	-	**1867,43**	-	-	-	-	-	-	-	-
31	93291	-	-	**1977,00**	-	-	-	-	-	-	-	-
32	99372	-	-	**2072,41**	-	-	-	-	-	-	-	-

Tabela 5.11. Resultados CEC para Duas Cópias de Divisores Restoring

Para os somadores (Rca x Rca, Tabela 5.13; Clad x Clad, Tabela 5.14; Csad x Csad, Tabela 5.15), o Minisat se sobressaiu no Rca x Rca e Csa x Csa. Para o Clad x Clad o Siege foi o que obteve melhores resultados com o aumento do tamanho dos circuitos. Os resolvedores propostos não tiveram desempenhos satisfatórios nesses casos.

Resultados CEC para Duas Cópias de Divisores Non Restoring											B-D SpeedUp(%)	
Divisor Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
3	442	0,01	0,01	0,01	**0,00**	**0,00**	0,01	0,01	0,01	0,01	-100,00	50,01
4	755	0,10	0,04	0,04	0,04	0,04	**0,03**	0,05	**0,03**	**0,03**	0,02	-5,00
5	1152	0,53	0,34	0,41	0,40	1,35	3,46	0,50	0,40	**0,29**	-70,37	-75,19
6	1633	3,22	2,62	2,62	3,83	**1,56**	>10000	3,69	4,30	3,51	145,71	68,25
7	2198	37,99	16,39	33,42	19,03	**3,32**	-	11,54	10,13	13,78	473,80	394,03
8	2847	281,64	220,93	322,36	40,43	**5,39**	-	33,60	16,31	21,58	650,21	3999,60
9	3580	2326,34	1579,84	2067,72	41,01	21,91	-	100,36	**20,70**	59,97	87,19	7111,35
10	4397	>10000	>10000	6268,13	64,86	43,31	-	627,82	**41,97**	126,26	49,75	-
11	5298	-	-	>10000	90,97	70,91	-	2354,63	**70,09**	290,44	28,30	-
12	6283	-	-	-	**120,51**	166,05	-	4859,40	220,91	489,52	-27,43	-
13	7352	-	-	-	262,01	**169,04**	-	>10000	429,19	1149,66	55,00	-
14	8505	-	-	-	**217,23**	410,91	-	-	600,73	1824,10	-47,13	-
15	9742	-	-	-	**380,68**	745,42	-	-	1055,04	3015,70	-48,93	-
16	11063	-	-	-	**504,79**	3662,64	-	-	1538,61	7805,34	-86,22	-
17	12468	-	-	-	**680,90**	1997,22	-	-	3162,39	9778,57	-65,91	-
18	13957	-	-	-	**898,51**	2942,23	-	-	4449,38	7318,76	-69,46	-
19	15530	-	-	-	**1379,19**	6071,07	-	-	6775,96	>10000	-77,28	-
20	17187	-	-	-	**1636,41**	>10000	-	-	>10000	-	-	-
21	18928	-	-	-	**2171,93**	-	-	-	-	-	-	-
22	20753	-	-	-	**2935,48**	-	-	-	-	-	-	-
23	22662	-	-	-	**4226,85**	-	-	-	-	-	-	-
24	24655	-	-	-	**5086,61**	-	-	-	-	-	-	-
25	26732	-	-	-	**5953,82**	-	-	-	-	-	-	-
26	28893	-	-	-	**8800,34**	-	-	-	-	-	-	-
27	31138	-	-	-	>10000	-	-	-	-	-		

Tabela 5.12. Resultados CEC para Duas Cópias de Divisores Non Restoring

Resultados CEC para Duas Cópias de Somadores Ripple Carry											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
128	5271	0,35	0,16	**0,15**	0,36	0,82	5,97	1,32	0,80	0,84	-56,09	-81,10
256	10519	1,64	**0,61**	0,64	1,53	4,85	35,00	7,62	4,61	4,60	-68,42	-87,33
384	15767	4,81	1,71	**1,70**	4,89	11,08	128,07	28,14	11,68	11,24	-55,85	-84,55
512	21015	10,51	**2,76**	3,49	10,67	21,82	322,05	49,72	22,16	22,14	-51,09	-87,33
640	26263	16,81	**3,42**	6,00	15,22	37,08	565,45	97,12	37,37	37,19	-58,95	-90,77
768	31511	25,45	**4,97**	8,43	25,38	56,61	1141,72	142,80	56,76	56,98	-55,17	-91,22
896	36759	40,40	**8,78**	12,36	30,65	78,81	1653,27	228,10	79,65	78,46	-61,11	-88,86
1024	42007	46,66	**10,71**	15,24	46,60	113,33	2334,46	338,74	119,85	108,65	-58,88	-90,55
1152	47255	65,61	**15,53**	18,76	86,39	150,10	4002,78	479,78	143,93	143,21	-42,45	-89,65
1280	52503	101,78	**19,11**	22,98	81,05	191,71	5404,01	587,30	207,55	188,39	-57,72	-90,03
1408	57751	104,82	**17,75**	29,79	134,81	229,68	7847,08	1151,86	248,06	238,16	-41,31	-92,27
1536	62999	121,64	**23,12**	38,54	434,06	302,20	>10000	1098,47	300,84	283,80	43,63	-92,35
1664	68247	154,84	**32,47**	45,44	161,42	350,69	-	1640,24	394,21	340,99	-53,97	-90,74
1792	73495	188,30	**28,02**	48,88	247,25	421,90	-	2296,96	436,07	417,11	-41,40	-93,36
1920	78743	216,82	**34,47**	53,61	273,00	503,77	-	2171,78	528,37	478,17	-45,81	-93,16
2048	83991	237,85	**49,51**	66,36	294,31	587,36	-	2176,90	637,71	571,43	-49,89	-91,57

Tabela 5.13. Resultados CEC para Duas Cópias de Somadores Ripple Carry

O BerkMin-D, que possui o método DBManag do Minisat, foi o que teve os resultados mais interessantes entre os resolvedores propostos, principalmente devido aos resultados obtidos para os multiplicadores Wallace e Dadda Tree. O BerkMin-D2 obteve bons resultados, ganhando em alguns momentos do BerkMin-D e até resolvendo mais instâncias do que este, como em Array x Dadda, Array x Wall, e Cla x Wall. O BerkMin* para os casos Array x Dadda e Array x Wall mostrou uma tendência de melhoria no tempo de execução com o aumento do tamanho dos circuitos, comparado

Resultados CEC para Duas Cópias de Somadores Carry LookAhead											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
32	16149	1,22	0,51	**0,49**	0,58	0,60	40,23	0,86	0,55	0,66	-3,96	-16,39
64	106005	9,79	47,58	**8,72**	26,19	13,01	2081,86	16,62	11,20	13,37	101,29	265,70
96	335125	97,14	555,05	**34,10**	204,15	72,40	>10000	91,43	52,17	75,11	181,99	666,67
128	769045	200,57	1494,83	**71,14**	646,60	248,09	-	372,64	184,62	268,52	160,63	502,54
160	1473301	674,12	*	**150,21**	3447,91	815,65	-	995,11	406,28	640,52	322,72	-
192	2513429	1188,37	-	**294,28**	9778,01	1577,11	-	2342,36	956,43	1498,98	520,00	-
224	3954965	4031,51	-	**453,10**	-	2858,90	-	4070,97	1963,63	2743,96	-	-
256	5863445	4005,46	-	**781,69**	-	5994,76	-	8368,11	3218,57	4962,23	-	-

Tabela 5.14. Resultados CEC para Duas Cópias de Somadores Carry LookAhead

Resultados CEC para Duas Cópias de Somadores Carry Save											B-D SpeedUp(%)	
Tam	**# Cl**	**Rsat**	**Minisat**	**Siege**	**Berkmin**	**B-D**	**B-DeD**	**B-De2D**	**B***	**B-D2**	**Berkmin**	**Minisat**
32	1429	0,02	0,02	**0,01**	**0,01**	0,05	0,13	0,07	0,04	0,05	-77,77	-66,67
64	2837	0,08	0,05	**0,04**	0,07	0,19	0,48	0,28	0,20	0,19	-62,36	-74,19
96	4245	0,16	0,10	**0,07**	0,21	0,44	1,33	0,67	0,53	0,44	-51,83	-76,38
128	5653	0,31	0,19	**0,14**	0,40	0,92	4,38	1,11	0,93	0,93	-56,52	-79,89
160	7061	0,60	0,30	**0,26**	0,70	1,35	6,16	2,01	1,63	1,43	-48,14	-77,63
192	8469	0,87	0,49	**0,41**	1,14	2,07	11,54	3,16	2,66	2,20	-44,95	-76,29
224	9877	1,30	0,66	**0,58**	1,54	3,10	19,45	4,71	3,55	3,29	-50,36	-78,70
256	11285	2,05	0,82	**0,78**	2,19	4,79	25,76	6,73	5,46	4,84	-54,29	-82,80
288	12693	2,52	1,19	**1,09**	3,38	6,12	30,44	8,87	7,19	6,87	-44,76	-80,60
320	14101	3,00	1,48	**1,36**	4,03	7,81	44,26	11,95	9,45	8,04	-48,38	-81,05
352	15509	3,96	**1,83**	1,86	5,68	9,94	43,97	14,28	12,10	10,78	-42,84	-81,57
384	16917	5,54	**1,93**	2,09	6,90	12,54	76,38	18,30	15,49	13,96	-44,97	-84,59
416	18325	5,24	**2,43**	2,56	8,28	15,15	80,03	20,75	18,46	16,13	-45,35	-84,00
448	19733	7,26	**2,92**	3,16	10,07	18,61	116,98	24,70	21,05	18,96	-45,90	-84,34
480	21141	9,69	**3,28**	3,74	13,87	21,72	105,56	31,77	25,72	22,72	-36,14	-84,91
512	22549	8,29	**3,74**	4,24	12,84	29,74	195,80	34,25	31,16	30,30	-56,83	-87,44

Tabela 5.15. Resultados CEC para Duas Cópias de Somadores Carry Save

a todos os resolvedores propostos e também ao BerkMin.

Era de se esperar que o BerkMin* fosse obter resultados mais próximos ao BerkMin. Uma possível explicação para essa diferença é a utilização da função monotônica crescente na escolha da próxima variável (Seção 3.6.1). Esta funcionalidade ainda não foi implementada. Outro possível fator são os detalhes, como parâmetros ou aspectos da implementação, que não foram descritos nos artigos. Provavelmente nunca se descobrirá quais são, pois o BerkMin é de código fechado.

Para avaliar a questão de erros de *cache*, apontada como justificativa da primeira hipótese[1], foi feita a análise dos erros da *cache* e quantidade de acessos para o BerkMin e o BerkMin-D, utilizando-se a ferramenta *cachegrind* do aplicativo *Valgrind* [Valgrind, 2010] para o multiplicador Wall x Wall e para o divisor Nrdivider x Nrdiviver. Os resultados podem ser vistos nas Figuras 5.1 e 5.2. Apesar do BerkMin possuir uma porcentagem maior de *misses*, ele possui uma menor quantidade de acessos para o Nrdivider. Observe o gráfico de números absolutos de erros de *cache* da Figura 5.2. Em 14x14 bits, o número de Misses de *cache* L1 é igual para o BerkMin-D e o Berk-

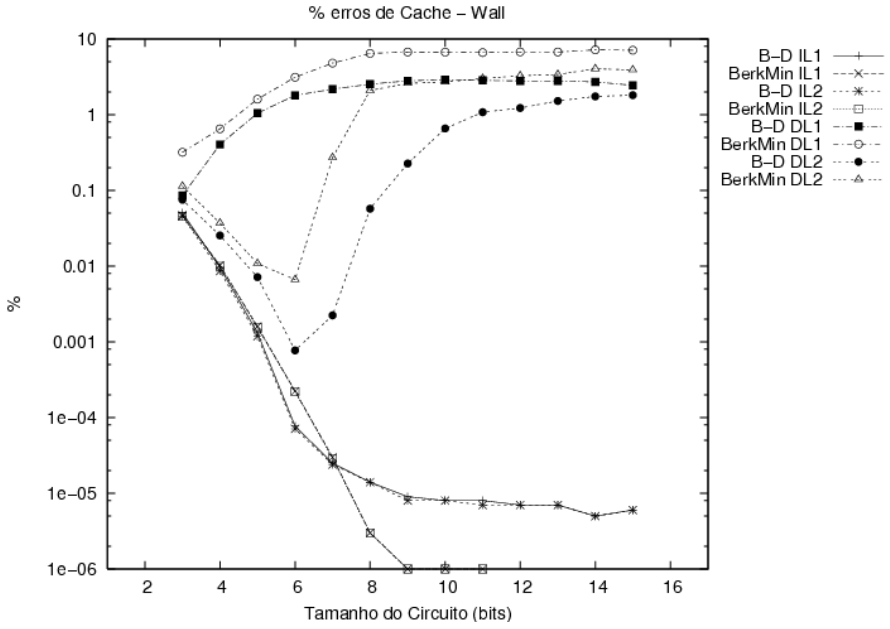
[1]Veja a Enumeração 5.1.

Min. Porém para o último, o número de erros de *cache* L2 é menor do que para o BerkMin-D. Logo o tempo de verificação para o BerkMin é menor. No caso 15x15, o número de Misses L1 do BerkMin-D passa a ser maior que o número de Misses L1 do BerkMin. Já no Wall x Wall, o BerkMin-D possui menor quantidade absoluta de *misses* a partir do 7x7 bits. Um resultado interessante é que a porcentagem de erros de *cache* do BerkMin L1 é sempre maior do que a do BerkMin-D, tanto para o Nrdivider quanto para o Wall.

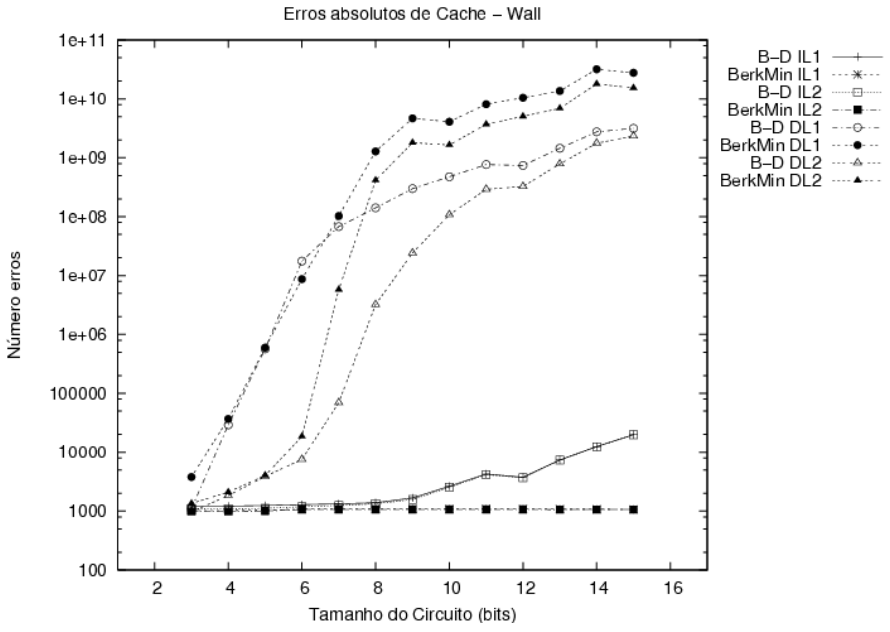
A partir das idéias do artigo [Selman et al., 1996], em que a razão do número de cláusulas pelo número de variáveis versus o número de chamadas recursivas geradas é analisada, gráficos da razão do número de cláusulas por variáveis para o número de bits de entrada do circuito foram criados[2]. Através do número de bits da entrada é possível verificar o tempo de execução para cada resolvedor. Os gráficos representando a quantidade de cláusulas pelo número de variáveis e a razão cláusulas por variáveis pelo número de bits de entrada são mostrados nas Figuras 5.3[3], 5.4 e 5.5; para os somadores, divisores e multiplicadores, respectivamente. Não foi encontrada nenhuma relação entre a razão Cláusulas x Variáveis e o tempo de execução para os multiplicadores, uma vez que, por exemplo, o BerkMin-D gasta menos tempo resolvendo o Wall x Wall que o Array x Dadda (com o aumento do circuito, razão menor que Wall x Wall) ou o Array x Cla (com o aumento do circuito, razão maior que Wall x Wall). Por sua vez, o Dadda x Dadda gasta menos tempo que o Array x Cla, mesmo tendo razão maior ou igual a esse. Nos divisores, a razão variáveis x cláusulas do Rdivider tem um crescimento menor que o do Nrdivider, que pode ter influenciado no tempo de execução do BerkMin e do Siege. O Siege não resolve nem a 11x11 bits do Nrdivider, mas foi o que apresentou melhores resultados para o Rdivider. Para os somadores, o Rca e o Csad tem razão constante, enquanto o Clad aumenta com o aumento do número de bits. O Clad realmente tem os maiores tempos de verificação, conforme a Tabela 5.14. Não foi encontrada uma razão em que os tempos começassem a cair de forma consistente, com o aumento ou diminuição da razão, como foi encontrado no artigo [Selman et al., 1996], para nenhum caso. Os resultados encontrados podem ser explicados pelo fato que todas as instâncias do problema são UNSAT, e não há o aspecto 50% de chance de ser SAT ou UNSAT, encontrada no artigo [Selman et al., 1996].

[2]É importante ressaltar que o artigo [Selman et al., 1996] baseia sua análise em instâncias K-SAT randômicas.

[3]Nesse caso, para a razão, o eixo Y tem escala logarítmica.

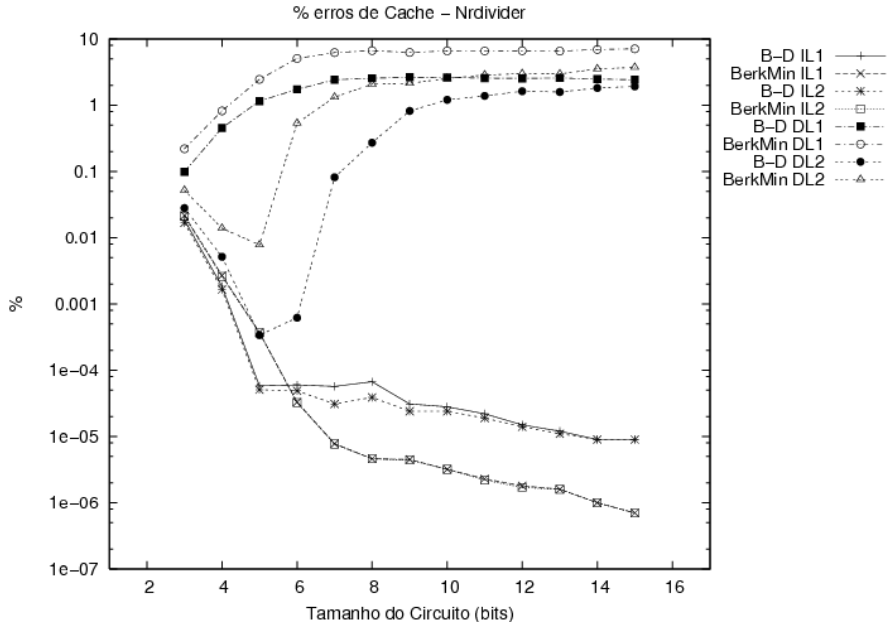


(a) Gráfico Tamanho x % erros de Cache

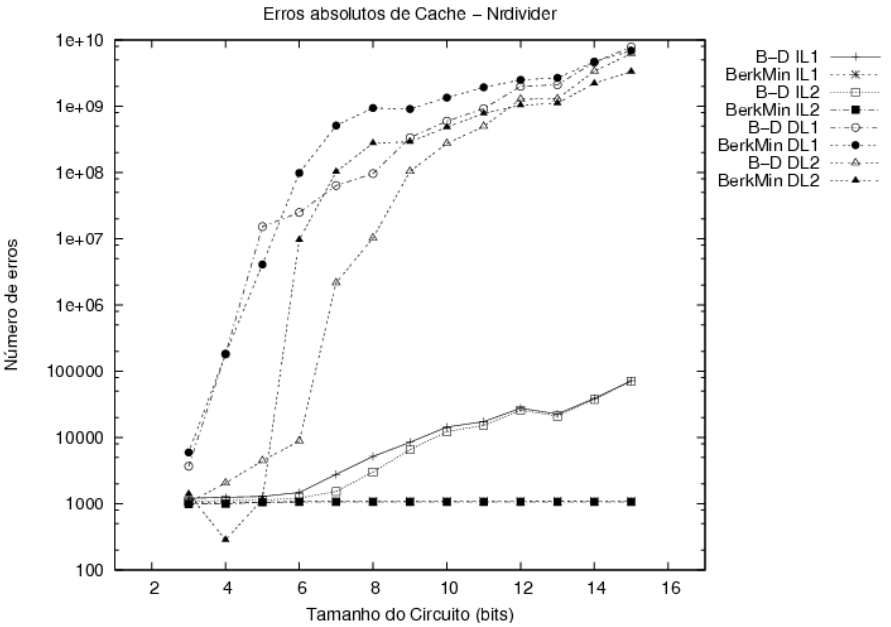


(b) Gráfico Tamanho x Número absoluto de erros de cache

Figura 5.1. Gráficos de Cache Misses para CEC Wall x Wall

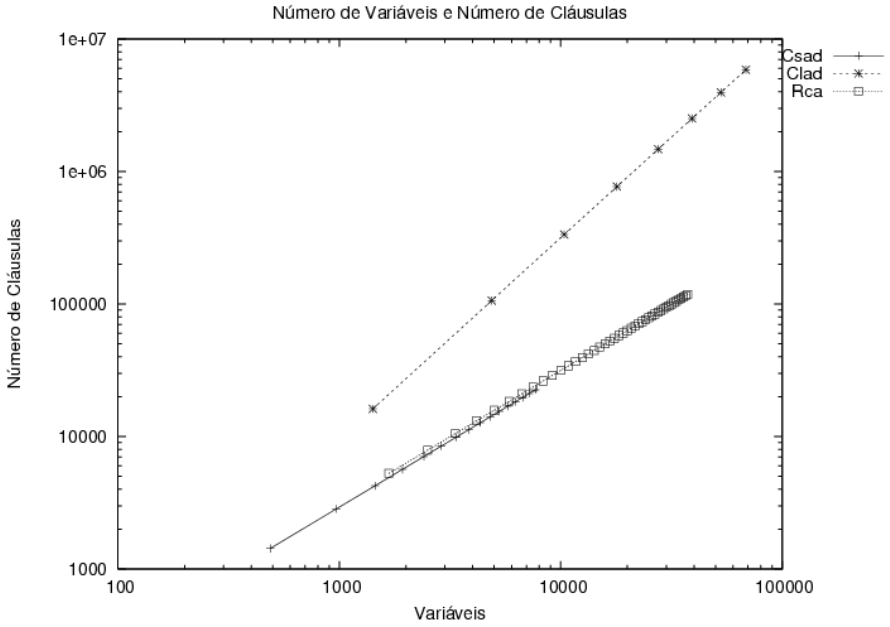


(a) Gráfico Tamanho x % erros de Cache

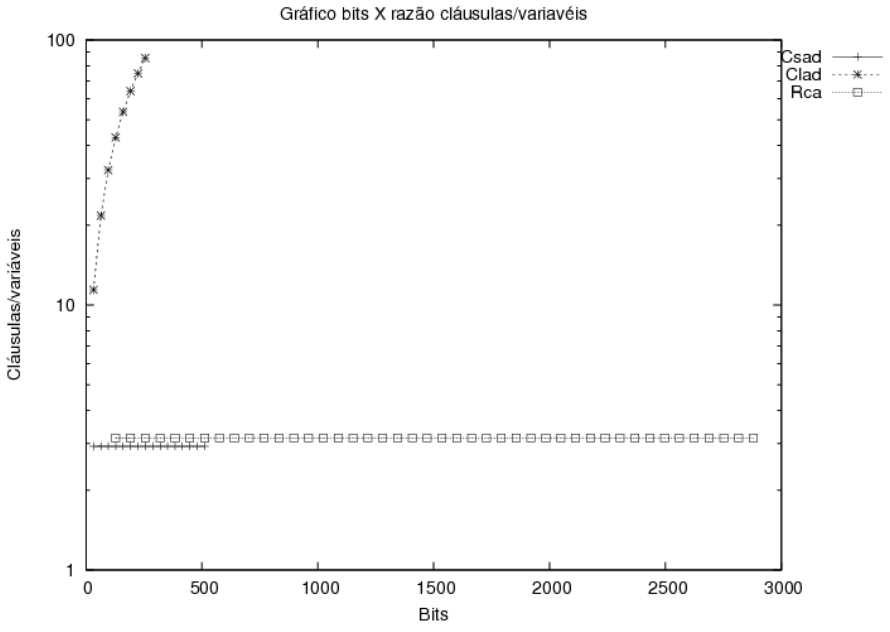


(b) Gráfico Tamanho x Número absoluto de erros de cache

Figura 5.2. Gráficos de Cache Misses para CEC Nrdivider x Nrdivider

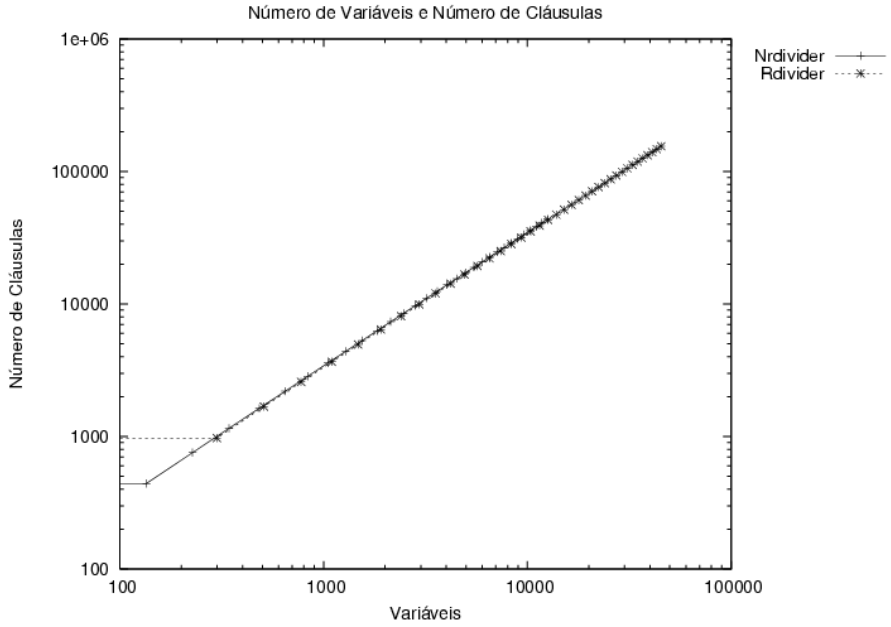


(a) Gráfico de Variáveis x Número de Cláusulas

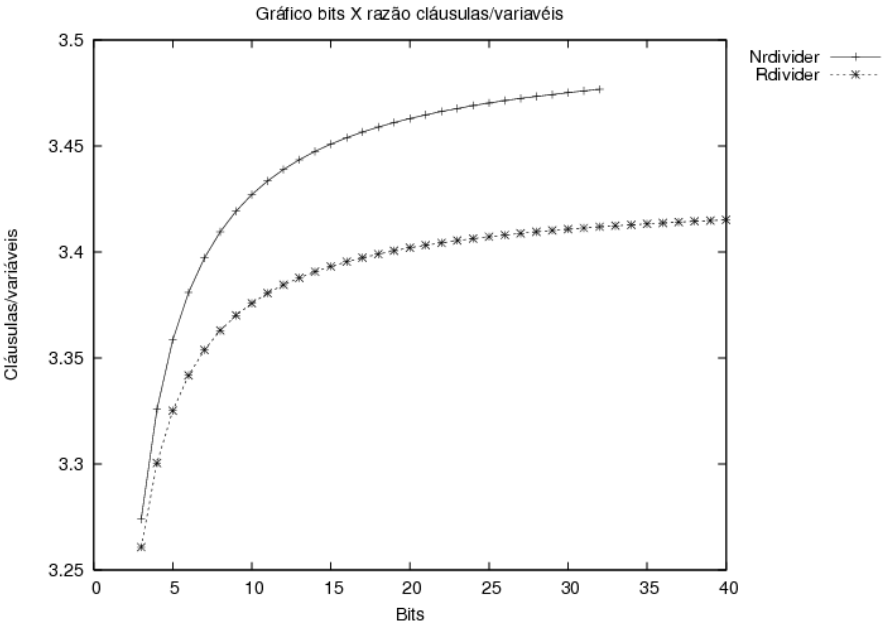


(b) Gráfico bits x Cláusulas/Variáveis

Figura 5.3. Gráficos dos Somadores

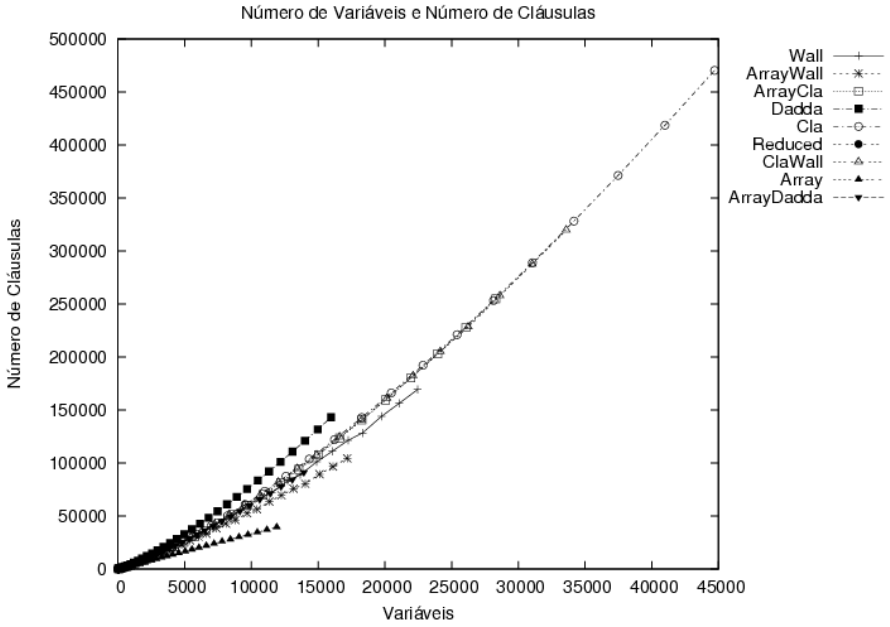


(a) Gráfico de Variáveis x Número de Cláusulas

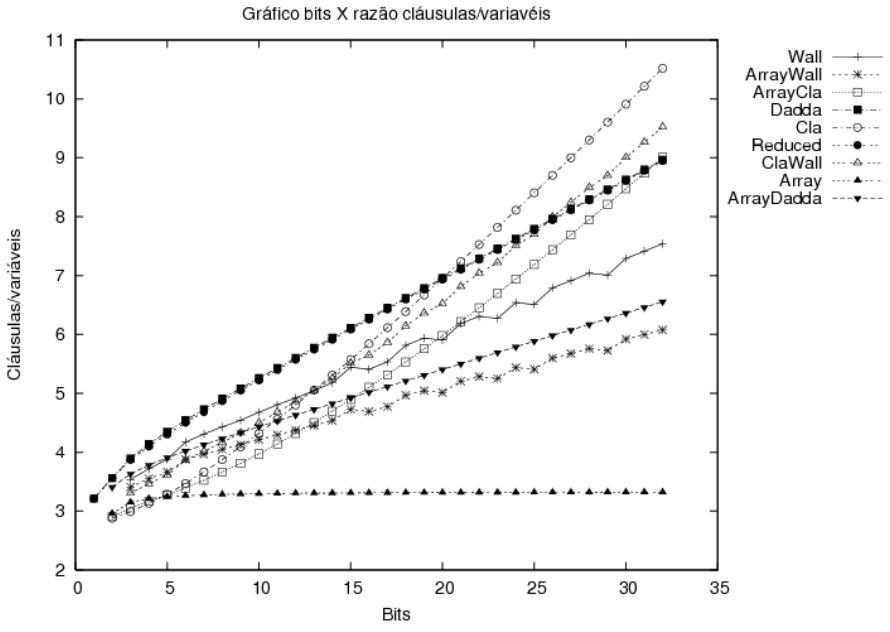


(b) Gráfico bits x Cláusulas/Variáveis

Figura 5.4. Gráficos dos Divisores



(a) Gráfico de Variáveis x Número de Cláusulas



(b) Gráfico bits x Cláusulas/Variáveis

Figura 5.5. Gráficos dos Multiplicadores

Capítulo 6

Conclusão e Trabalhos Futuros

6.1 Conclusão

Este trabalho apresentou duas contribuições. A primeira é uma nova abordagem na construção de resolvedores de SAT através de um resolvedor modular. A partir dessa contribuição, através da simples escolha das heurísticas utilizadas, melhores tempos de execução foram alcançados para algumas instâncias de CEC.

A proposta de um resolvedor modularizado tornou mais fácil e rápida a codificação e a combinação dos módulos, permitindo grande possibilidade de testes. O BerkMin-D, com uma técnica mais agressiva de retirada de cláusulas aprendidas da base de dados, mostrou-se eficiente, principalmente para os casos dos multiplicadores Dadda Tree, Wallace Tree e Reduced Tree. As outras propostas não tiveram resultados tão bons. Porém, o BerkMin-D2 teve resultados melhores que o BerkMin-D, para alguns circuitos multiplicadores com pouca similaridade, indicando que a mudança em tempo de execução da heurística DBManag é promissora nesses casos. O BerkMin* também se sobressaiu comparado aos outros resolvedores propostos para os multiplicadores com poucas similaridades.

Um fator interessante é a eficiência do Minisat para os circuitos somadores. A abordagem modular é benéfica nas situações em que diferentes conjuntos de heurísticas são melhores quando aplicadas a diferentes tipos de resolvedores, já que o mesmo código é utilizado em qualquer situação. Desta forma, o código fica cada vez mais robusto.

A análise da porcentagem de erros de *cache* e do total de acessos mostrou que o BerkMin-D possui menor porcentagem de *cache misses* que o BerkMin. Logo, evidencia que as cláusulas menores aprendidas, realmente diminuíram a porcentagem de erros de *cache*. Porém o BerkMin tem um número total de *cache misses* menor para o caso Nrdivider, o que faz com que ele seja mais rápido, evidenciando que, ou a heurística

de decisão do BerkMin, ou as cláusulas mantidas na base de cláusulas aprendidas do BerkMin, ajudam a resolver esse tipo de problema mais rapidamente.

Quanto à adoção da análise dos erros de *cache*, para a verificação dos padrões de acessos à *cache*[1], é importante ressaltar que esta análise pode ocasionar ganhos de desempenho consideráveis. Como a implementação do BerkMin é fechada não é possível saber qual estrutura de dados é efetivamente utilizada e nem como ela é implementada, em comparação à implementada pelo resolvedor modular.

A razão de cláusulas/variáveis não mostrou relação com o tempo de execução para os multiplicadores. Para os divisores e os somadores essa razão indicou o aumento do tempo de execução para os circuitos com maiores razões. Não foi encontrada uma razão em que os tempos começassem a cair de forma consistente com o aumento da razão, como foi encontrado no artigo [Selman et al., 1996], para nenhum caso. Isto, possivelmente, deve-se ao fato que todas as instâncias testadas são UNSAT.

6.2 Trabalhos futuros

Pode-se reimplementar métodos de outros resolvedores e testar o desempenho alcançado. Uma modificação importante que pode ser feita no método DBManag do Minisat é a mudança do ponto onde a atividade passa a ser considerada para remoção da cláusula. Já que o método do BerkMin trabalha diferentemente entre os $\frac{1}{16}$ iniciais e os $\frac{15}{16}$, pode-se mudar esse ponto de metade para $\frac{1}{16}$.

Deve-se também implementar a multiplicação da função monotônica durante a fase de escolha da próxima variável, o que inclui aspectos estruturais à busca. A heurística J-Frontier, que também utiliza aspectos estruturais em resolvedores baseados na representação em circuitos, encontrada em [Wu et al., 2007], também deve ser analisada e adaptada para o resolvedor baseado em FNC.

A análise de *cache misses* deve ser estendida para os outros resolvedores. Dessa forma pode-se comparar o ganho de performance de heurísticas que potencialmente escolhem melhores variáveis (menos acessos) ou de heurísticas que tem menor porcentagem de erros (possivelmente com cláusulas aprendidas menores, ou estruturas de dados com padrões de acesso mais bem definidos). Diferentes tipos de organização e associatividade da *cache* devem ser testados. O impacto do uso de comandos de pré-carregamento de endereços, específicos da arquitetura do processador, também deve ser avaliado.

[1]Relacionada a diferentes estruturas de dados, ou diferentes organizações da *cache*.

A versão 2.0 beta do Minisat, utilizada como base nesta dissertação, possui uma classe SimpSolver, que faz simplificações lógicas utilizando uma implementação dos métodos do SatELite. Esta classe deve ser estuda e integrada ao resolvedor modular, a fim de verificar uma possível melhora na performance.

Referências Bibliográficas

[Altera, 2009] Altera, C. (2009). *QUARTUS II INTRODUCTION FOR VERILOG USERS*. Altera Coporation.

[Andrade et al., 2008a] Andrade, F.; Silva, L. & Fernandes, A. (2008a). Improving sat-based combinational equivalence checking through circuit preprocessing. In *Computer Design, 2008. ICCD 2008. IEEE International Conference on*, pp. 40–45.

[Andrade, 2008] Andrade, F. V. (2008). *Contribuições para o problema de Verificação de Equivalência de Circuitos Combinacionais*. PhD thesis, Universidade Federal de Minas Gerais.

[Andrade et al., 2008b] Andrade, F. V.; Silva, L. M. & Fernandes, A. O. (2008b). 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.

[Ben-Sasson et al., 2004] Ben-Sasson, E.; Impagliazzo, R. & Wigderson, A. (2004). Near optimal separation of tree-like and general resolution. *Combinatorica*, 24(4):585--603.

[Biere et al., 1999] Biere, A.; Cimatti, A.; Clarke, E. M. & Zhu, Y. (1999). Symbolic model checking without bdds. In *TACAS '99: Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pp. 193--207, London, UK. Springer-Verlag.

[Brien & Malik, 2006] Brien, C. & Malik, S. (2006). Understanding the dynamic behavior of modern dpll sat solvers through visual analysis. pp. 49–50.

[Clarke et al., 2003] Clarke, E.; Kroening, D. & Yorav, K. (2003). Behavioral consistency of c and verilog programs using bounded model checking. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pp. 368--371, New York, NY, USA. ACM.

[Clarke et al., 1999] Clarke, E. M.; Grumberg, O. & Peled, D. A. (1999). *Model Checking*. The MIT Press, 1 edição.

[Cook, 1971] 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.

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

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

[de Campos Lynce Faria, 2004] de Campos Lynce Faria, M. I. C. (2004). *Propositional Satisfiability: Techniques, Algorithms and Applications*. PhD thesis, Universidade Tecnica de Lisboa.

[DIMACS, 1993] DIMACS (1993). Np hard problems: Maximum clique, graph coloring, and satisfiability - the second dimacs implementation challenge. `ftp://dimacs.rutgers.edu/pub/challenge/sat/doc/`.

[Eén & Sörensson, 2003] Eén, N. & Sörensson, N. (2003). An extensible sat-solver. In Giunchiglia, E. & Tacchella, A., editores, *SAT, volume 2919 of Lecture Notes in Computer Science*, pp. 502–518. Springer.

[Ganai et al., 2002] Ganai, M. K.; Ashar, P.; Gupta, A.; Zhang, L. & Malik, S. (2002). Combining strengths of circuit-based and cnf-based algorithms for a high-performance sat solver. In *DAC '02: Proceedings of the 39th annual Design Automation Conference*, pp. 747--750, New York, NY, USA. ACM.

[Giunchiglia & Tacchella, 2004] Giunchiglia, E. & Tacchella, A., editores (2004). *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*. Springer.

[Goldberg & Novikov, 2002] 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 & Novikov, 2003] Goldberg, E. & Novikov, Y. (2003). Equivalence checking of dissimilar circuits. In *IWLS-2003 12th International Workshop on Logic and Synthesis*.

[Goldberg & Novikov, 2007] Goldberg, E. & Novikov, Y. (2007). Berkmin: A fast and robust sat-solver. *Discrete Applied Mathematics*, 155(12):1549 – 1561. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing.

[Gomes et al., 2008] Gomes, C. P.; Kautz, H.; Sabharwal, A. & Selman, B. (2008). *Satisfiability Solvers, Handbook of Knowleadge Representation*, chapter 2. ELSEVIER.

[Gomes et al., 1998] Gomes, C. P.; Selman, B. & Kautz, H. (1998). Boosting combinatorial search through randomization. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pp. 431--437, Menlo Park, CA, USA. American Association for Artificial Intelligence.

[Gupta & Ashar, 1998] 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.

[Gupta et al., 2003] Gupta, A.; Ganai, M.; Wang, C.; Yang, Z. & Ashar, P. (2003). Learning from bdds in sat-based bounded model checking. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pp. 824--829, New York, NY, USA. ACM.

[Hutter et al., 2007] Hutter, F.; Babic, D.; Hoos, H. H. & Hu, A. J. (2007). Boosting verification by automatic tuning of decision procedures. In *FMCAD '07: Proceedings of the Formal Methods in Computer Aided Design*, pp. 27--34, Washington, DC, USA. IEEE Computer Society.

[Innovation & Lab, 2002] Innovation & Lab, T. R. (2002). Sat live! `http://www.satlive.org/`.

[Kang & Park, 2003] Kang, H.-J. & Park, I.-C. (2003). Sat-based unbounded symbolic model checking. In *DAC '03: Proceedings of the 40th annual Design Automation Conference*, pp. 840--843, New York, NY, USA. ACM.

[Kautz & Selman, 2007] Kautz, H. & Selman, B. (2007). The state of sat. *Discrete Applied Mathematics*, 155(12):1514 – 1524. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing.

[KhudaBukhsh et al., 2009] KhudaBukhsh, A. R.; Xu, L.; Hoos, H. H. & Leyton-Brown, K. (2009). Satenstein: automatically building local search sat solvers from components. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*, pp. 517--524, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Knot, 2007] Knot (2007). Rsat 2.0: Sat solver description. Technical report, Automated Reasoning Group, Computer Science.

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

[Malik et al., 2001] Malik, S.; Zhao, Y.; Madigan, C. F.; Zhang, L. & Moskewicz, M. W. (2001). Chaff: Engineering an efficient sat solver. *Design Automation Conference*, 0:530–535.

[Marques-Silva & Sakallah, 1999] Marques-Silva, J. P. & Sakallah, K. A. (1999). Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521.

[Marques-Silva & Sakallah, 2000] Marques-Silva, J. P. & Sakallah, K. A. (2000). Boolean satisfiability in electronic design automation. In *DAC '00: Proceedings of the 37th conference on Design automation*, pp. 675--680, New York, NY, USA. ACM.

[McMillan, 1992] McMillan, K. L. (1992). *Symbolic model checking: an approach to the state explosion problem*. PhD thesis, Pittsburgh, PA, USA.

[McMillan, 2002] McMillan, K. L. (2002). Applying sat methods in unbounded symbolic model checking. In *CAV '02: Proceedings of the 14th International Conference on Computer Aided Verification*, pp. 250--264, London, UK. Springer-Verlag.

[Mishchenko, 2009] Mishchenko, A. (2009). *ABC: A System for Sequential Synthesis and Verification*. Electrical Engineering and Computer Sciences UC Berkley, http://www.eecs.berkeley.edu/ alanmi/abc/.

[Molitor & Mohnke, 2004] Molitor, P. & Mohnke, J. (2004). *Equivalence Checking of Digital Circuits: Fundamentals, Principles, Methods*. Springer, 1 edição.

[Moore, 2006] Moore, G. E. (2006). Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff. *Solid-State Circuits Newsletter, IEEE*, 20(3):33–35.

[Ozguner et al., 2001] Ozguner, F.; Marhefka, D.; DeGroat, J.; Wile, B.; Stofer, J. & Hanrahan, L. (2001). Teaching future verification engineers: the forgotten side of logic design. In *DAC '01: Proceedings of the 38th conference on Design automation*, pp. 253--255, New York, NY, USA. ACM.

[Paruthi & Kuehlmann, 2000] Paruthi, V. & Kuehlmann, A. (2000). Equivalence checking combining a structural sat-solver, bdds, and simulation. volume 0, p. 459, Los Alamitos, CA, USA. IEEE Computer Society.

[Prasad et al., 2005] Prasad, M. R.; Biere, A. & Gupta, A. (2005). A survey of recent advances in sat-based formal verification. *International Journal on Software Tools for Technology Transfer (STTT)*, 7(2):156--173.

[Ryan, 2002] Ryan, L. (2002). Efficient algorithms for clause-learning sat solvers. Master's thesis, Simon Fraser University.

[Samulowitz & Memisevic, 2007] Samulowitz, H. & Memisevic, R. (2007). Learning to solve qbf. In *AAAI'07: Proceedings of the 22nd national conference on Artificial intelligence*, pp. 255--260. AAAI Press.

[SATLIB, 2008] SATLIB (2008). Satlib. `http://www.satlib.org/`.

[Selman et al., 1996] Selman, B.; Mitchell, D. G. & Levesque, H. J. (1996). Generating hard satisfiability problems. *Artif. Intell.*, 81(1-2):17--29.

[Stallman & Sussman, 1977] Stallman, R. M. & Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artif. Intell.*, 9(2):135–196.

[Stephan et al., 1996] Stephan, P.; Brayton, R. & Sangiovanni-Vincentelli, A. (1996). Combinational test generation using satisfiability. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 15(9):1167–1176.

[Valgrind, 2010] Valgrind (2010). *Valgrind 3.5.0*. Valgrind Developers, http://valgrind.org/.

[Vieira, 2008] Vieira, N. J. (2008). *Apostila de Lógica Proposicional*. Belo Horizonte, MG.

[Wu et al., 2007] Wu, C.-A.; Lin, T.-H.; Lee, C.-C. & Huang, C.-Y. (2007). Qutesat: A robust circuit-based sat solver for complex circuit structure. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE '07*, pp. 1–6.

[Wu & Hsiao, 2008] Wu, W. & Hsiao, M. (2008). Sat-based state justification with adaptive mining of invariants. In *Test Conference, 2008. ITC 2008. IEEE International*, pp. 1–10.

[Xu et al., 2008] Xu, L.; Hutter, F.; Hoos, H. H. & Leyton-Brown, K. (2008). Satzilla: portfolio-based algorithm selection for sat. *J. Artif. Int. Res.*, 32(1):565--606.

[Zhang, 1997] 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.

[Zhang et al., 2001] Zhang, L.; Madigan, C. F.; Moskewicz, M. H. & Malik, S. (2001). Efficient conflict driven learning in a boolean satisfiability solver. In *ICCAD '01: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, pp. 279--285, Piscataway, NJ, USA. IEEE Press.

[Zhang & Malik, 2003] Zhang, L. & Malik, S. (2003). Validating sat solvers using an independent resolution-based checker: Practical implementations and other applications. In *DATE '03: Proceedings of the conference on Design, Automation and Test in Europe*, p. 10880, Washington, DC, USA. IEEE Computer Society.