PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós-Graduação em Informática

PUC-SAT: um solucionador híbrido baseado nos algoritmos de Stalmarck e DPLL

Flávio Márcio de Moraes e Silva

Belo Horizonte
2010

Flávio Márcio de Moraes e Silva

PUC-SAT: um solucionador híbrido baseado nos algoritmos de Stalmarck e DPLL

Dissertação apresentada ao Programa de Pós-Graduação em Informática como requisito parcial para obtenção do Grau de Mestre em Informática pela Pontifícia Universidade Católica de Minas Gerais.

Orientador: Mark Alan Junho Song

Belo Horizonte
2010

FICHA CATALOGRÁFICA

Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

S586p

Silva, Flávio Márcio de Moraes e
PUC-SAT: um solucionador híbrido baseado nos algoritmos de Stalmarck e DPLL / Flávio Márcio de Moraes e Silva . - Belo Horizonte, 2010.
70f. : il.

Orientador: Mark Alan Junho Song
Dissertação (Mestrado) - Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-graduação em Informática.
Bibliografia.

1. Algoritmos - Teses. 2. Lógica simbólica e matemática. I. Song, Mark Alan Junho. II. Pontifícia Universidade Católica de Minas Gerais. III. Título.

CDU: 681.3.056

Bibliotecário: Fernando A. Dias - CRB6/1084

PUC Minas
Programa de Pós-graduação em Informática

FOLHA DE APROVAÇÃO

PUC-SAT: um solucionador híbrido baseado nos algoritmos de Stalmark e DPLL

FLÁVIO MÁRCIO DE MORAES E SILVA

Dissertação defendida e aprovada pela seguinte banca examinadora:

Prof. Mark Alan Junho Song - Orientador (PUC Minas)
Doutor em Ciências da Computação

Prof. Antônio Otávio Fernandes (UFMG)
Doutor em Informática

Prof. Romanelli Lodron Zuim - (PUC Minas)
Doutor em Ciências da Computação

Profa. Raquel Aparecida de Freitas Mini (PUC Minas)
Doutora em Ciências da Computação

Belo Horizonte, 19 de agosto de 2010.

DEDICATÓRIA

Dedico este trabalho aos meus pais Antônio Márcio e Celeida.

AGRADECIMENTOS

Agradeço a inicialmente a Deus, por me dar o dom da vida e me iluminar nos momentos de dificuldade.

Aos meus pais pelo apoio e afeto, sem eles não teria chegado até aqui.

A minha irmã Natália e meu cunhado Bruno pela amizade e companheirismo de sempre.

A todos os meus tios, tias e primos pelo apoio. Em especial a minha tia Deneise, que sempre me apoiou em todos os momentos de minha vida.

A minha namorada Fabiana pela paciência, incentivo e apoio ao longo da realização deste trabalho.

Aos meus amigos que estão sempre muito presentes em minha vida com companheirismo, alegria e entusiasmo.

À Pontifícia Universidade Católica de Minas Gerais por me propiciar os melhores mentores e a melhor infra-estrutura durante a realização deste trabalho.

Aos colegas de mestrado pelas contribuições técnicas, científicas e motivacionais que ajudaram na elaboração do trabalho. Em especial ao colega Fernando Augusto que trabalhou conosco durante todo o projeto.

Ao professor Mark Alan pelas imprescindíveis orientações ao longo do projeto e pela paciência durante os momentos de dificuldade.

Aos meus alunos que fomentam minha dedicação aos estudos e à pesquisa.

Enfim a todos aqueles que me ajudaram direta ou indiretamente ao longo da realização deste projeto.

RESUMO

O problema de Satisfabilidade (SAT), problema este NP-completo, é objeto de estudo de várias áreas da ciência da computação. Para a resolução do mesmo, diversas alternativas foram e são freqüentemente propostas. O presente trabalho envolve o estudo dos algoritmos de Stalmarck e DPLL. Com base nestes trabalhos foi implementado o solucionador híbrido PUC-SAT, que mescla conceitos destes dois algoritmos. Nesta ferramenta podem ser encontradas implementações de código aberto para os métodos propostos por Stalmarck, motivando assim pesquisas relacionadas a este algoritmo. A partir desta solução verifica-se a utilização do método proposto por Stalmarck para o tratamento de problemas NÃO SAT, assim como algumas propostas de adaptação do algoritmo para avaliar problemas satisfazíveis. Tal solução apresentou resultados interessantes para instâncias industrias NÃO SAT. As técnicas e a metodologia envolvida na criação do solucionador são apresentadas no presente trabalho, assim como as análises relativas à ferramenta.

Palavras-chave: Lógica Matemática, Lógica Proposicional, Satisfabilidade, SAT, Stalmarck, DPLL

ABSTRACT

The problem of satisfiability (SAT), this problem NP-complete, has been studied in various areas of computer science. For the same resolution, several alternatives were and are often proposed. This work involves the study of algorithms Stalmarck and DPLL. Based on these works was implemented hybrid solver PUC-SAT, which merges the concepts of these two algorithms. This tool can be found in open source implementations for the methods proposed by Stalmarck, motivating research related to this algorithm. From this solution there is to use the method proposed by Stalmarck to treat problems NOT SAT, as well as some proposals to adapt the algorithm to evaluate satisfiability problems. This solution presented interesting results for industrial instances NOT SAT. The techniques and methodology involved in creating the solver are presented in this work, as well as reviews of the tool.

Key-words: Mathematical Logic, Propositional Logic, Satisfiability, SAT, Stalmarck, DPLL

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BCP *Boolean Constraint Propagation*

BDD *Binary Decision Diagram*

CNF *Conjunctive Normal Form*

DP Algoritmo de Davis e Putnam

DPLL Algoritmo de Davis-Putnam-Logemann-Loveland

EDA *Eletronic Design Automation*

EI Expressão de Implicação

FN Forma Normal

FNC Forma Normal Conjuntiva

FND Forma Normal Disjuntiva

IJCAR *International Joint Conference on Automated Reasoning*

NP Não Polinomial

P Polinomial

SAT Satisfabilidade

VI Variáveis de Implicação

VO Variáveis Originais

VPI *Vital Processor Interlocking*

VSIDS *Variable State Independent Decaying Sum*

SUMÁRIO

1 INTRODUÇÃO

No século XX muitos foram os problemas computacionais destacados como importantes durante o processo evolutivo da Ciência da Computação. Dentre estes pode-se destacar o problema da satisfabilidade, também chamado de SAT, conhecido por sua importância teórica sendo que este foi o primeiro a ser provado NP-Completo (COOK, 1971). Na segunda metade do século XX muitas foram as alternativas propostas para resolução deste problema, com destaque para o algoritmo proposto por Davis, Putnam, Logemann e Loveland (1962); referenciado neste trabalho pela sigla DPLL, base para a construção da maioria das soluções atuais. Em virtude da aplicabilidade do problema em diversas áreas como: ciência da computação, eletrônica, engenharia, biologia, matemática e muitas outras; intensa tem sido a pesquisa para a evolução dos solucionadores SAT.

1.1 Objetivo do Trabalho

Para realização do presente trabalho foram estudadas algumas propostas da literatura para a resolução de problemas envolvendo expressões proposicionais. Apesar de grande parte das ferramentas SAT atuais ainda se basearem na abordagem proposta por Davis e Loveland (1962), existem outras propostas para resolução de problemas similares com resultados expressivos. Dentre essas destaca-se o algoritmo para verificação de tautologia de Sheeran e Stalmarck (1998), algoritmo este patenteado, de código fechado e comprovadamente eficiente para instâncias industriais (STALMARCK, 1989b).

O presente trabalho dedica-se a implementação de uma ferramenta de código aberto que utiliza conceitos dos trabalhos de Sheeran e Stalmarck (1998), Stalmarck (1989a) e também de ferramentas evoluídas a partir do algoritmo DPLL. Devido a existência de poucas ferramentas SAT, que não sejam apenas evoluções do algoritmo DPLL, tomou-se como motivação a possibilidade de construção de um solucionador híbrido para o tratamento do problema SAT, motivando assim pesquisas relacionadas a soluções alternativas para o mesmo.

1.2 O Problema SAT

Para melhor entendimento do problema SAT, alguns conceitos básicos serão abordados a seguir.

- Variáveis proposicionais ou variáveis *booleanas* são variáveis que podem assumir somente os valores "*verdadeiro*" ou "*falso*".
- Uma fórmula ou expressão proposicional é composta por variáveis proposicionais, operadores lógicos $\wedge$ e $\vee$ (referentes aos operadores *AND* e *OR*), operador unário $\neg$ (para representação lógica da negação) e parênteses que alteram a precedência dos operadores. Tome como exemplo a expressão $\neg\ (x_1) \vee (x_1 \wedge x_2)$.
- Um literal é representado por uma variável proposicional, x_i, ou sua negação $\neg\ x_i$.
- Literais podem ser agrupados definindo cláusulas. Estas são definidas pela disjunção de literais. Sendo C_n uma cláusula, onde "n" é o tamanho da cláusula, $C_n = \neg\ x_1 \vee x_2 \vee \neg\ x_3 \vee ... \vee x_n$ é dita satisfeita se um literal presente na conjunção encerra o valor 1 (Verdadeiro) e não satisfeita se todos os literais encerram o valor 0 (Falso). Quando uma cláusula possui somente um literal, por exemplo C = x_1, esta é chamada cláusula unitária.
- Uma expressão *booleana* f é dita estar na forma normal conjuntiva (FNC) quando definida por uma conjunção de cláusulas sendo $f = c_1 \wedge c_2 \wedge c_3 \wedge ... \wedge c_n$ e quando são satisfeitas as restrições:
  - Contém, quando muito, os conectivos $\neg$, $\wedge$, $\vee$;
  - O operador $\neg$ não pode ser repetido (como $\neg\neg$) e não tem alcance sobre $\wedge$ e $\vee$ (isto é, só incide sobre variáveis proposicionais)
  - O operador $\vee$ não tem alcance sobre $\wedge$ (isto é, não há componentes do tipo x$\vee$(q$\wedge$r))

  Note que f é satisfeita se e somente se todas as cláusulas assumem o valor 1 (Verdadeiro), e não satisfeita se uma ou mais cláusulas assumirem 0 (Falso).
- Diz-se que uma expressão booleana é satisfeita ou satisfazível se existe pelo menos uma atribuição de valores às variáveis que tornem a expressão na FNC verdadeira.

A partir dos conceitos apresentados define-se o problema SAT como:

"*O problema da satisfabilidade (problema SAT) consiste em determinar se uma fórmula, na FNC, é satisfazível ou não.*"

Este problema pertence à classe dos problemas de decisão, ou seja, problemas que admitem dois tipos de solução "*sim*" ou "*não*". Para qualquer instância do problema SAT, um solucionador deve responder "*sim*" caso a instância seja satisfazível e "*não*" caso contrário. Basicamente, a tarefa de um solucionador SAT é encontrar uma atribuição de valores para as variáveis, presentes na instância, de forma a satisfazer todas as cláusulas. Encontrando tal atribuição resolveu-se o problema.

1.3 Aplicações para o problema SAT

Inúmeros problemas podem ser configurados como um problema de satisfabilidade. Alguns exemplos desta possibilidade já foram explorados em estudos relacionados a: Ciência da Computação, Matemática, Biologia, Eletrônica e outras. As subseções seguintes apresentam exemplos que motivam a pesquisa em satisfabilidade.

1.3.1 Teoria da Complexidade

A classe P (*Polynomial-time Algorithms*) reúne os problemas que podem ser resolvidos por algoritmos deterministas em tempo polinomial. Um algoritmo é dito não-determinista quando, diante de duas ou mais alternativas, é capaz de adivinhar a alternativa que leva a solução. Sendo a classe NP (*Nondeterministic Polinomial Time Algorithms*) aquela que engloba os problemas que podem ser resolvidos por algoritmos não-deterministas em tempo polinomial. Como algoritmos deterministas são apenas um caso especial de algoritmos não-deterministas, pode-se concluir que a classe P está contida em NP. O que não se sabe é se P é igual a NP, ou seja, se existem algoritmos polinomiais deterministas para todos os problemas de NP.

Um problema de decisão X é completo em NP ou NP-Completo, se X pertence a NP e qualquer outro problema presente em NP é redutível a X. Um problema é redutível a outro problema, se existe um algoritmo polinomial em máquina determinista que transforma qualquer instância "a" do primeiro em uma instância "b" do segundo. Encontrando-se uma resposta para "b" encontra-se também a resposta para "a". Cook (1971) demonstrou que existe uma redução de qualquer problema NP para o problema NP-Completo SAT,

objeto de estudo do presente trabalho. Encontrada a solução polinomial para um problema da classe NP-Completo, comprovar-se-ia a hipótese de que P = NP (P=NP, 2009).

1.3.2 EDA - Electronic Design Automation

Pelo fato de circuitos eletrônicos serem facilmente mapeáveis em um problema de satisfabilidade, grande é a utilização de ferramentas SAT na simulação dos mesmos. Devido aos avanços na área de satisfabilidade, esta tem se tornado objeto de estudo cada vez maior por parte da comunidade EDA. Vários são os trabalhos relacionando o problema SAT à automação de projetos eletrônicos, como pode ser observado no artigo de Marques-Silva e Skallah (2000). Dentre estes destacam-se os relacionados a: testes (LARRABEE, 1992) (STHEPAN; SANGIOVANNI-VINCENTELLI, 1996), verificação de microprocessadores (VELEV; BRYANT, 2001), verificação de equivalência entre circuitos (MARQUES-SILVA; GLASS, 1999), verificação de circuitos (SHEERAN; BORäLV, 1997), verificação formal (BIERRE ALESSANDRO CIMATTI; ZHU, 1999) (MACMILLAN, 2002), dentre outros.

Um trabalho interessante envolvendo a utilização do algoritmo de Stalmarck pode ser encontrado em Groote, Vlijmen e Koorn (1995). Este trabalho diz respeito a construção do equipamento *Vital Processor Interlocking (VPI)*, responsável pela prevensão de desastres, da estação de trens Hoorn-Kersenboogerd na Holanda.

1.3.3 Verificação Formal

A verificação formal é uma técnica criada na década de 80, pelos pesquisadores Emerson e Clarke (1980), Clarke e Emerson (1981), Queille e Sifakis (1982) e Clarke e Sistla (1986). Esta técnica consiste na verificação de sistemas computacionais, através da configuração dos mesmos em modelos de estado finito concorrentes. Tais modelos são representados por uma estrutura denominada *Binary Decision Diagram*(BDD). Qualquer problema computacional pode ser configurado como uma expressão proposicional. O BDD é uma forma de representar uma expressão proposicional através de um grafo acíclico dirigido, em que os vértices representam as variáveis (vértices não-terminais) e as soluções do problema (vértices terminais); enquanto as arestas representam as possíveis atribuições de valores para as variáveis, no caso de expressões proposicionais os valores possíveis são 0 ou 1.

Por se tratarem de técnicas que lidam com verificação e com expressões proposicionais, forte é a ligação entre a satisfabilidade e a verificação formal de modelos. Podendo BDD's

serem convertidos em problemas de satisfabilidade e vice-versa. Assim sendo, muitas são as ferramentas ligadas a verificação formal que utilizam heurísticas e técnicas provenientes das soluções para satisfabilidade, dentre as quais destaca-se o NuSMV (CIMATTI E. M. CLARKE; TACCHELLA, 2002).

1.3.4 Provadores de Teoremas

Um antigo desafio da Matemática consiste na necessidade de se provar a validade de teoremas. Sendo a Ciência da Computação e a Teoria da Prova disciplinas derivadas da lógica matemática é razoavel imaginar a utilização de sistemas computacionais para a realização de provas automáticas de teoremas. Neste contexto muitos tem sido os esforços nos últimos anos para criação e evolução dos sistemas desta natureza. As principais publicações deste seguimento de pesquisa podem ser encontradas nos anais da conferência *IJCAR - International Joint Conference on Automated Reasoning* realizada anualmente (IJCAR, 2010). Dentre as diversas classes de provadores, destacam-se:

- Verificadores de tautologias - dada uma proposição, procura-se inferir se esta é uma tautologia, verificando a existência de atribuições que a tornam inválida;
- Verificadores de provas - o provador lê uma sequência de fórmulas lógicas e verifica se cada uma pode ser derivada da anterior, dado seu conjunto de axiomas;
- Focados em um único teorema - o provador é projetado para verificar um determinado teorema. Em geral, tratam-se de teoremas notoriamente difíceis e a solução consiste na busca por simplicações que tornem a verificação possível.

O algoritmo de Sheeran e Stalmarck (1998), que é um dos objetos de estudo do presente trabalho, pode ser colocado na primeira classe de verificadores. Por outro lado o problema da satisfabilidade pode ser visto como um teorema, sendo os solucionadores SAT verificadores focados na sua solução. Muitas são as técnicas e estudos compartilhados entre a comunidade SAT e as comunidades que trabalham com provadores de teoremas. Não é incomum encontrar solucionadores SAT sendo utilizados em provadores de teoremas (PROF et al., 2004), (BUCH; HILLENBRAND, 1996).

1.4 Organização do Trabalho

Este trabalho está dividido da seguinte forma. No capítulo 2 são apresentados os algoritmos utilizados como referencial teórico para desenvolvimento do trabalho. No capítulo 3 são discutidas as adaptações nos algoritmos de Stalmarck e DPLL para a abordagem híbrida. No capítulo 4 são apresentadas as análises relacionadas à ferramenta PUC-SAT e no capítulo 5 as propostas de trabalhos futuros e as conclusões obtidas. No apêndice encontram-se conceitos básicos sobre algebra das proposições que ajudam na compreensão do texto.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os conceitos e algoritmos utilizados para realização deste trabalho.

2.1 Competicão SAT

A competição SAT (COMPETITION, 2009) é um concurso de ferramentas SAT realizado anualmente, desde 2002, com o intuito de avaliar novos solucionadores em relação às soluções que já fazem parte do estado da arte. A idéia é propiciar a evolução e discussão sobre as heurísticas utilizadas nas ferramentas, bem como fomentar a evolução dos *benchmarks* utilizados. Alguns dos solucionadores SAT e os *benchmarks* utilizados nesse trabalho foram escolhidos com base nessa competição, que é sub-dividida em três classes de instâncias:

- *Random*: criadas randômicamente.
- *Crafted*: construídas.
- *Application*: problemas industriais reais transformados em instâncias SAT.

Para cada classe existem três categorias de problemas: satisfazíveis, não satisfazíveis e indeterminados. Podendo os solucionadores competir por classe, por categoria ou por classe e categoria específicas. Sendo premiados os três melhores por classe e categoria.

2.2 Algoritmo DPLL

Em 1960 Davis e Putnam propuseram o algoritmo DP, basicamente uma versão preliminar do algoritmo DPLL capaz de resolver o problema SAT, mas com deficiência em relação à utilização exponencial de espaço (DAVIS; PUTNAM, 1960).

A partir de análises e melhorias no algoritmo DP, em (1962) Davis, Logemann e Loveland propuseram o algoritmo DLL que resolveria o problema da utilização exponencial de memória. Mais tarde o algoritmo DLL viria a ser renomeado para DPLL, em reconhecimento a contribuição dos quatro autores na criação do mesmo (DAVIS; LOVELAND, 1962).

Como basicamente um problema SAT consiste na busca de atribuição de valores às variáveis de uma expressão proposicional que a torne verdadeira, um tratamento intuitivo para o problema é dispor as variáveis presentes na expressão na forma de uma árvore binária de decisão percorrendo-se os caminhos à procura da solução. Considerando-se uma expressão com n variáveis tem-se 2^n caminhos distintos a serem percorridos. O algoritmo DPLL consiste em um procedimento de busca em profundidade com retrocesso, através do universo dos caminhos possíveis. O mecanismo de resolução deste algoritmo baseia-se nos seguintes critérios:

- BCP (*Boolean Constraint Propagation*) - se houver uma cláusula com um só literal, ou seja uma cláusula unitária, este literal terá que fazer parte da resposta final. As cláusulas que contêm o literal podem ser eliminadas e as que contêm a negação do literal podem ser simplificadas.

- Eliminação de literal puro - as cláusulas que contêm literais puros podem ser eliminadas do problema, um literal é dito puro se aparece somente em fase positiva ou somente negado em toda a fórmula.

- Busca com retrocesso: sendo Φ a fórmula na FNC, quando não for mais possível aplicar o BCP e a eliminação de literais puros em Φ, escolha arbitrariamente uma variável "x" e execute o algoritmo para $\Phi \wedge (x)$. Se a chamada recursiva retornar SAT, retorne SAT. Caso contrário execute o algoritmo para $\Phi \wedge (\neg x)$. Se a chamada recursiva retornar SAT, retorne SAT, senão retorne Não-SAT.

No algoritmo 1 apresenta-se um esboço do método DPLL. Sendo Φ a fórmula na FNC e Π o conjunto de cláusulas atribuidas como verdade, que está inicialmente vazio:

Algorithm 1 Método DPLL

função DPLL(Φ, Π)
(1) se todas as cláusulas de Φ forem verdadeiras
então retorne verdadeiro;
(2) se alguma cláusula de Φ for falsa
então retorne falso;
(3) se ocorrer uma cláusula unitária c em Φ
então retorne DPLL(atribuição(c,Φ), Π ∧ c);
(4) se ocorrer um literal puro c em Φ
então retorne DPLL(atribuição(c,Φ), Π ∧ c);
(5) c := escolhaliteral(Φ);
(6) retorne DPLL(atribuição(c,Φ), Π ∧ c) ou DPLL(atribuição(¬c,Φ), Π ∧ ¬c);
fim

No algoritmo DPLL destacam-se 3 mecanismos fundamentais:

- Decisão: consiste na escolha das variáveis e na atribuição de valor em cada estágio do processo de busca. Este mecanismo pode ser observado nos passos 3, 4, 5 e 6.
- Dedução: consiste na resolução da expressão com base nas atribuições de valores para as variáveis. Este mecanismo pode ser observado nos passos 3 - aplicação do BCP e 4 - regra do literal puro.
- Diagnóstico e Retrocesso: situação em que o mecanismo de dedução não se aplica embora a expressão ainda não esteja resolvida. Este mecanismo pode ser observado nos passos 5 e 6.

É importante ressaltar que a maioria das ferramentas SAT atuais são evoluções do algoritmo DPLL. Tais evoluções são, basicamente, diferentes implementações para os mecanismos descritos acima.

2.3 Solucionadores Baseados em DPLL

2.3.1 ZChaff

Sendo uma ferramenta baseada nos princípios do DPLL, o solucionador ZChaff (ZCHAFF-HOMEPAGE, 2010) incorpora e em alguns casos melhora técnicas propostas nas ferramentas SATO (ZHANG, 1997), GRASP (MARQUES-SILVA; SAKALLAH, 1999) e principalmente Chaff (MOSKEWICZ C. F. MADIGAN; MALIK, 2001). Com destaque para os seguintes aspectos:

- Melhoria na estrutura de dados que envolve as listas *Head and Tail* propostas por (ZHANG, 1997), substituindo-a por uma vigilância através de literais sentinelas.
- Utilização da técnica VSIDS(*Variable State Independent Decaying Sum*) proposta na ferramenta SATO.
- Adota-se o processo de aprendizagem por conflitos e backtracking não cronológico discutidos e aplicados na ferramenta GRASP.
- Utilização da técnica de *restarts* que, baseada no aprendizado de clausuas proposto em GRASP, evita que o algoritmo SAT se perca em partes não relevantes do espaço de procura.

2.3.2 *RSat*

Também derivado do DPLL, o solucionador RSat (RSAT-HOMEPAGE, 2010) herda muitas das técnicas introduzidas pelo MiniSat (E´EN; RENSSON, 2005) e o ZChaff (ZCHAFF-HOMEPAGE, 2010). Apresentando três melhorias:

- Uma proposta baseada no mecanismo de conflitos dirigidos do ZChaff e no regime de cláusulas de aprendizagem do MiniSat, que reduz o tempo gasto no BCP.
- Uma nova técnica de gerência da fila de implicações baseada no trabalho (LEWIS M. D. T.; BECKER, 2004), que permite o resolvedor focar melhor em sub-problemas.
- E uma heurística de decisão que foi concebida para evitar a repetição de trabalhos já feitos, em situações de retrocesso.

2.4 Algoritmo de Stalmarck

O algoritmo de Stalmarck tem por objetivo a verificação de tautologia sobre uma expressão proposicional, diferente do algoritmo DPLL se caracteriza por ser de código fechado e também por se basear em uma busca em largura no universo de soluções (SHEERAN; STALMARCK, 1998), (STALMARCK, 1989b). Devido a esta característica, uma importante etapa preliminar se faz necessária para que uma instância SAT, originalmente na forma FNC, seja transformada em uma instância passível de execução pelo algoritmo de Stalmarck. Neste trabalho denomina-se tal etapa como método de transformação ou simplesmente transformação.

2.4.1 Transformação

O método de transformação consiste na conversão de uma expressão proposicional originalmente no formato FNC, para um formato que será referenciado por expressão de implicação (EI). Tal transformação ocorre em complexidade linear (STALMARCK, 1989a). O método consiste na transformação de uma expressão na forma FNC, utilizando a aplicação recorrente das seguintes transformações:

1) $A \vee B$ para $\neg A \rightarrow B$

2) $A \wedge B$ para $\neg (A \rightarrow \neg B)$

3) $\neg\neg A$ para A

4) $\neg A$ para $A \rightarrow 0$

Este trabalho refere-se a estas quatro regras como regras de transformação. Outra importante notação utilizada na quarta regra de transformação e que será utilizada no restante do texto é a representação do valor V (verdadeiro) através do valor 1 (um) e o valor F (falso) pelo valor 0 (zero).

A utilização das regras de transformação resulta em uma expressão que contém somente variáveis, parênteses e o conectivo de implicação $\rightarrow$. O exemplo seguinte demonstra a utilização das regras de transformação.

Considere a expressão na FNC, $(a \vee b) \wedge (\neg a)$. Aplicando as regras de transformação, tem-se:

- Transformação 4: $(a \vee b) \wedge (a \rightarrow 0)$
- Transformação 1: $(\neg a \rightarrow b) \wedge (a \rightarrow 0)$
- Transformação 4: $((a \rightarrow 0) \rightarrow b) \wedge (a \rightarrow 0)$
- Transformação 2: $\neg(((a \rightarrow 0) \rightarrow b) \rightarrow \neg(a \rightarrow 0))$
- Transformação 4: $\neg(((a \rightarrow 0) \rightarrow b) \rightarrow ((a \rightarrow 0) \rightarrow 0))$
- Transformação 4: $((((a \rightarrow 0) \rightarrow b) \rightarrow ((a \rightarrow 0) \rightarrow 0)) \rightarrow 0)$

Uma expressão do tipo $A \rightarrow B$, pode ser vista como $C \equiv A \rightarrow B$ ou $C \equiv \neg A \vee B$. Onde a nova variável C representa o resultado de $A \rightarrow B$. No método de Stalmarck cada

expressão do tipo C $\equiv$ A $\rightarrow$ B é representada por uma estrutura denominada "Tripla", que possui o formato (C,A,B).

Como resultado da transformação, tem-se:

((((a$\rightarrow$ 0) $\rightarrow$ b) $\rightarrow$ ((a $\rightarrow$ 0) $\rightarrow$ 0)) $\rightarrow$ 0)

Convertendo a expressão de implicação para uma representação de "Triplas", obtem-se:

- $X_1 \equiv$ (a $\rightarrow$ 0)
- Substituindo: ((X_1 $\rightarrow$ b) $\rightarrow$ (X_1 $\rightarrow$ 0)) $\rightarrow$ 0)
- $X_2 \equiv$ (X_1 $\rightarrow$ 0)
- $X_3 \equiv$ (X_1 $\rightarrow$ b)
- Substituindo: (($X_3$$\rightarrow$$X_2$) $\rightarrow$ 0)
- $X_4 \equiv$ (X_3 $\rightarrow$ X_2)
- Substituindo: (X_4 $\rightarrow$ 0)
- $X_5 \equiv$ (X_4 $\rightarrow$ 0)
- Resultando na expressão: X_5

Note que agora o problema (a $\vee$ b) $\wedge$ ($\neg$a) é representado pelas triplas (X_5,X_4,0), (X_4,X_3,X_2), (X_3,X_1,b), (X_2,X_1,0) e (X_1,a,0).

Após o método de transformação, as variáveis presentes na instância original são mantidas no problema e novas variáveis são adicionadas, para representar as implicações, gerando-se assim um grande número de novas variáveis. No caso do exemplo acima, foram geradas as novas variáveis X_1, X_2, X_3, X_4 e X_5. Este trabalho refere-se às variáveis originais do problema através do termo "variável original" e às novas variáveis criadas por "variável de implicação". Analisando o exemplo acima percebe-se que as variáveis de implicação na verdade não são variáveis do problema, mas apenas representações do conectivo lógico $\rightarrow$ (implicação).

Observando o processo de transformação pode-se perceber que a variável X_5 equivale a última substituição e consequentemente representa toda a expressão original (a$\vee$b)$\wedge$($\neg$a). O aumento no número de variáveis poderia ser considerado um aspecto negativo no processo proposto por Stalmarck, mas será mostrado a seguir que o processo de resolução

da nova estrutura possui características que tornam o processo eficiente, para expressões proposicionais que são tautologias. É importante ressaltar que a expressão de implicação gerada, após a utilização do método de transformação, é equivalente à expressão original na FNC. O que pode ser provado pelo princípio de equivalência exposto no capítulo anterior.

2.4.2 Resolução

O método de resolução proposto por Stalmarck consiste em uma prova por absurdo, onde se parte do pré-suposto (hipótese) de que a fórmula é equivalente à falso (F). Caso a hipótese seja confirmada tem-se que a expressão avaliada não é uma tautologia, em contrapartida caso seja encontrado um absurdo tem-se que a expressão é uma tautologia.

Durante o método de resolução da expressão de implicação um absurdo é alcançado quando se encontra, o que Stalmarck descreve como: tripla terminal ou tripla absurdo. As triplas terminais são (1,1,0), (0,X,1) e (0,0,Y).

Tomando como exemplo a expressão a $\rightarrow$ (b $\rightarrow$ a), equivalente às triplas $X_1 \equiv$ b$\rightarrow$a, $X_2 \equiv$a$\rightarrow X_1$. Note que a variável X_2 representa o resultado da expressão inicial logo assume-se que esta variável é Não SAT, ou seja igual a 0 (zero). A partir desta decisão uma série de ações são disparadas no universo de triplas, que compõem o problema. Essas ações são consequências da aplicação das sete regras de Stalmarck expostas a seguir, que são referenciadas neste trabalho também por "regras de decisão/dedução" ou simplesmente "sete regras".

Tomando-se por base uma tripla genérica (R,X,Y) onde R$\equiv$X$\rightarrow$Y ou R$\equiv\neg$X$\vee$Y. As seguintes decisões podem ser tomadas:

As sete regras de Stalmarck

Regra 1: (0,X,Y) $\Rightarrow$ X = 1 e Y = 0

Regra 2: (R,X,1) $\Rightarrow$ R = 1

Regra 3: (R,0,Y) $\Rightarrow$ R = 1

Regra 4: (R,1,Y) $\Rightarrow$ R = Y

Regra 5: (R,X,0) $\Rightarrow$ R = $\neg$X

Regra 6: (R,R,Y) $\Rightarrow$ R = 1 e Y = 1

Regra 7: (R,X,X) $\Rightarrow$ R = 1

As sete regras apresentadas acima são responsáveis pelo processo de decisão ou escolha dos valores a serem atribuídos às variáveis do problema e por consequência também realizam o processo de dedução ou resolução do problema. Conforme o exposto por Sheeran e Stalmarck (1998), o processo de resolução assim como as sete regras expostas, foram criados a partir dos trabalhos de Gentzen (1969), Kleene (1967), Smullyan (1969), Beth (1955), Kanger (1957), Hintikka (1955), Shütte (1977), Mondadori (1989) e D'Agostino (1990) .

O exemplo a seguir ilustra a aplicação das mesmas. Considerando a expressão a→(b→a), representada pelas triplas (X_2,a,X_1) (X_1,b,a).

- assumindo que $X_2 = 0$ pode-se aplicar a regra R1 na tripla (X_2,a,X_1), assim sendo, tem-se que a = 1 e $X_1 = 0$;
- em decorrência da propagação acima chega-se na tripla (0,b,1), que é uma tripla terminal;
- logo pode-se deduzir que considerar a expressão do exemplo como sendo equivalente à falso (hipótese), é um absurdo. Ou seja, a expressão é uma tautologia.

2.4.3 Dilema

Note que as sete regras propostas por Stalmarck nem sempre podem ser aplicadas. Tome como exemplo, uma situação em que se detecta uma tripla do tipo (1,X,1). Para essa tripla não se sabe que valor atribuir para a variável X. Nestes casos tem-se o que Stalmarck descreve como dilema (MONDADORI, 1989), (D'AGOSTINO, 1990).

A tarefa a ser executada em um dilema é verificar os dois valores possíveis para a variável X, sendo S_1 o conjunto de atribuições geradas a partir do caminho onde X=1 e S_0 o conjunto de atribuições geradas a partir do caminho onde X=0. Caso obtenha-se um absurdo em ambos os caminhos a expressão é considerada uma tautologia. Caso a contradição seja encontrada somente em um dos caminhos, por exemplo S_1, são mantidas as atribuições feitas no outro caminho S_0. Para o caso em que o absurdo não seja encontrado em nenhum dos dois caminhos, é mantida a interseção dos caminhos S_1 e S_0, ou seja as atribuições que aconteceram tanto em um caminho quanto no outro (RICHARDS, 1998), (SYRF, 1999).

2.4.4 Saturação e Complexidade

Saturação é um termo relacionado à complexidade da instância do problema e também à extensão do método dilema. Existem problemas cuja solução pode ser encontrada, por meio apenas da execução recorrente das sete regras de decisão/dedução. Um exemplo de problema desta natureza é a→(b→a). Note que constatou-se que esta expressão é uma tautologia, sem a necessidade de execução do dilema. Problemas desta natureza são considerados 0-Saturação. Pode-se compreender 0-Saturação como uma classe de problemas, cuja resolução é dada através da avaliação de um único caminho de atribuições de valores às variáveis.

Problemas cuja solução não pode ser encontrada somente através das sete regras, ou seja utilizam o método dilema, são problemas pertencentes a uma das classes n-Saturação onde $1 \leq n \leq \infty$.

O termo n exposto acima diz respeito ao número de variáveis simultâneas levadas ao dilema. No exemplo dado na seção anterior levou-se somente uma variável "X" ao dilema, o que ocasionou a análise de 2^1 caminhos simultâneos, ou seja, quando trabalha-se com somente uma variável no dilema, analisar-se-á no máximo 2 caminhos, um quando X=0 e outro quando X=1. Problemas solucionáveis através deste método são da classe 1-Saturação.

Existem casos em que a solução para o problema não é encontrada através de 1-Saturação. Fazendo-se necessária a ampliação da regra dilema para se trabalhar com 2 variáveis simultâneas, por exemplo X e Y. Note que com 2 variáveis no dilema tem-se a análise de 2^2 caminhos simultâneos, referentes às diferentes possibilidades de atribuição de valores para as variáveis X e Y (combinações: 00,01,10,11).

O algoritmo de Stalmarck garante a solução para o problema ampliando-se sucessivamente o método dilema até n-Saturação, onde n é o número de variáveis presentes no problema. Segundo Sheeran e Stalmarck (1998), o algoritmo possui complexidade $O(n^{2k+1})$ sendo n o número de variáveis da instância e k o nível de saturação. Note que para a situação extrema em que k=n o algoritmo apresenta péssimo desempenho. Mas, segundo Stalmarck, na prática, a maioria dos problemas reais (classe industrial) são solucionáveis em até 2-Saturação. O que coloca o algoritmo como boa solução para esta classe de problemas.

Uma análise de complexidade mais detalhada sobre o algoritmo de Stalmarck, inclusive comparando-o ao método de resolução proposto em DPLL, pode ser encontrada no

trabalho de Nordström (2001).

2.5 Solucionador HeerHugo

Um trabalho realizado por Groote e Warners (2000), que também utiliza os conceitos dos algoritmos DPLL e Stalmarck, envolve o desenvolvimento do provador HeerHugo. Este provador utiliza as técnicas de transformação propostas por Stalmarck, para converter expressões proposicionais arbitrárias para o formato FNC. Após esta transformação, simplificações são feitas na expressão com base nas técnicas de decisão e dedução propostas por Davis e Loveland (1962), aliadas a técnicas relacionadas a BDD's. Quando simplificações não são mais possíveis utiliza-se o método de saturação proposto por Stalmarck para concluir o processo de resolução. O código fonte bem como resultados relacionados a este provador não foram encontrados para efeito de comparação com a ferramenta PUC-SAT.

Diferente do provador HeerHugo, a ferramenta PUC-SAT utiliza inicialmente as técnicas propostas por Davis e Loveland (1962) para simplificar uma expressão já no formato FNC. Posteriormente transforma-se a expressão resultante para o formato de implicações. Onde são aplicadas as técnicas de decisão e dedução propostas por Sheeran e Stalmarck (1998), acrescidas de algumas adaptações propostas no presente trabalho.

2.6 Mecanismos - DPLL x Stalmarck

Conforme explicado anteriormente verificou-se que a maioria dos solucionadores possuem basicamente os mecanismos de: decisão, dedução e análise, diagnóstico e retrocesso em conflitos. No caso do algoritmo de Stalmarck existe também o mecanismo de transformação.

A seguir serão mostradas algumas evoluções das soluções baseadas em DPLL, mais referenciadas na comunidade SAT ao longo dos anos para os mecanismos citados. E também as propostas do algoritmo de Stalmarck para resolver os mesmos problemas.

2.6.1 Decisão

Um dos pontos críticos no desempenho do solucionador SAT são as heurísticas de decisão ou, em outras palavras, a estratégia de escolha da próxima variável a ser atribuída e o valor a ser atribuído a essa variável. A ordem de escolha pode ser estática (pré-determinada) ou decidida dinamicamente de acordo com o atual estado da busca.

No decorrer dos anos várias heurísticas diferentes foram propostas por diferentes pesquisadores (ZHANG, 1997) (MARQUES-SILVA; SAKALLAH, 1999) (MOSKEWICZ C. F. MADIGAN; MALIK, 2001) (GOLDBERG; NOVIKOV, 2002).

As primeiras heurísticas como, por exemplo, buscar a variável com o máximo de ocorrências em cláusulas de tamanho mínimo, são consideradas heurísticas gulosas porque tentam gerar o maior número de implicações ou satisfazer a maioria das cláusulas. Elas são úteis para instâncias SAT aleatórias, mas normalmente não têm um desempenho destacado em instâncias estruturadas (BURO; BUNING, 1992).

Em um de seus trabalhos Zhang (1997), propôs uma heurística denominada Soma Decadente Independente do Estado da Variável (*Variable State Independent Decaying Sum - VSIDS*). VSIDS mantém uma pontuação para cada variável. Inicialmente a pontuação é simplesmente o número de ocorrências do literal na fórmula.

À medida que novas cláusulas que contêm uma variável são adicionadas ao banco de cláusulas, VSIDS aumenta a pontuação dessa variável por uma constante. Além disso, à medida que a busca progride, as pontuações são divididas por uma constante. A consequência desse mecanismo é que variáveis com maior pontuação tendem a ser aquelas presentes nas cláusulas adicionadas mais recentemente. A variável com maior pontuação é a escolhida pelo motor de dedução. Por ser independente do estado das variáveis o mecanismo é barato de se manter.

Os experimentos mostram que não só o mecanismo é competitivo, mas também utiliza apenas uma pequena porcentagem do tempo total de processamento. Essa heurística é usada, por exemplo, nos solucionadores derivados da ferramenta Chaff (MOSKEWICZ C. F. MADIGAN; MALIK, 2001).

Posteriormente foram propostas por Goldberg e Novikov (2002) evoluções no esquema VSIDS. A estratégia continua sendo a busca por variáveis que foram ativas recentemente. Entretanto, há diferença entre os dois esquemas: enquanto VSIDS cria sua pontuação de acordo com as ocorrências das variáveis, a nova abordagem estabelece a pontuação baseada em conflitos. Quando um conflito ocorre todos os literais que foram responsáveis pelo mesmo tem sua pontuação incrementada. A pontuação nesse esquema também é dividida por uma constante no decorrer do tempo, privilegiando as variáveis presentes nos conflitos mais recentes. Comparativamente o referido esquema é mais robusto em relação ao proposto em VSIDS. A heurística apresentada é utilizada em alguns solucionadores propostos ao longo dos últimos anos como o MiniSat e o Glucose (AUDEMARD; SIMON, 2009a), vencedor da competição SAT em 2009 na classe industrial NÃO-SAT.

No algoritmo de Stalmarck o mecanismo de decisão está embutido nas sete regras expostas na seção 2.4.2. Assim sendo, o valor a ser atribuído a uma variável depende de como as triplas, presentes na expressão original, se enquadram para serem mapeadas em uma das sete regras. A ordem de verificação das triplas pode alterar o resultado final, mas possivelmente devido ao fato do algoritmo ser de código fechado, não foram encontrados trabalhos propondo algum tipo de heurística desta natureza.

2.6.2 Dedução

O motor de dedução poda o espaço de buscas através de implicações geradas a partir da atribuição de uma variável. Vários mecanismos de dedução têm sido propostos. Entretanto, estudos comprovam que o mecanismo mais eficiente continua sendo a regra da cláusula unitária (DAVIS; LOVELAND, 1962). Todos os solucionadores SAT atuais derivados de DPLL, incluem a mesma no motor de dedução. Como o BCP normalmente é a etapa que ocupa a maior parte do tempo de processamento, sua implementação influi diretamente na eficiência do algoritmo.

Uma implementação simples e intuitiva para o BCP é manter contadores para cada cláusula. O solucionador GRASP, por exemplo, utiliza tal política (MARQUES-SILVA; SAKALLAH, 1999). Neste, cada cláusula mantém dois contadores, um para o número de literais que receberam o valor 1 e outro para os literais que receberam o valor 0. Cada variável mantém duas listas: uma com as cláusulas em que aparece como literal positivo e outra com as cláusulas em que aparece como literal negativo.

Quando uma variável é atribuída todas as cláusulas que contém essa variável, em uma das suas fases, tem seus contadores atualizados. Se o contador de literais de valor 0, de uma cláusula, contiver um valor igual ao número total de literais tem-se um conflito. Se o valor for menor em uma unidade tem-se uma cláusula unitária. O mecanismo é fácil de entender e implementar embora não seja o mais eficiente.

Uma abordagem interessante de BCP é utilizada no solucionador Chaff (MOSKEWICZ C. F. MADIGAN; MALIK, 2001). A solução é denominada vigilância por dois literais. A idéia é baseada no seguinte princípio: se uma cláusula tem n literais, então só será preciso se preocupar com a mesma quando n - 1 variáveis forem atribuídas com 0. Ou seja, só se aplica a regra da cláusula unitária quando o número de literais atribuídos com 0 passar de n - 2 para n - 1.

Na implementação, cada cláusula tem dois literais especiais que são denominados

literais de vigilância. Inicialmente nenhum literal de vigilância está atribuído. Quando um dos literais for atribuído com 0 uma das seguintes condições há de valer:

- Existe um literal na cláusula que não foi atribuído com 0 e não é o outro literal de vigilância. Então simplesmente substituí-se o literal de vigilância.
- Apenas o outro literal de vigilância não está atribuído. Então a cláusula é unitária.
- Apenas o outro literal de vigilância não está atribuído com 0. Mas, está atribuído com 1. A cláusula foi satisfeita. Não há nada mais a ser feito.
- Não há outro literal que não tenha sido atribuído com 0. Tem-se um conflito.

Este mecanismo apresenta duas grandes vantagens: é preciso examinar apenas cláusulas onde os literais de vigilância recebem o valor 0 e não é preciso alterar os literais de vigilância durante um eventual retrocesso. Uma vez que foram os últimos a serem atribuídos com 0, ou perderão a atribuição ou receberão 1 como valor.

No algoritmo de Stalmarck o processo de dedução é feito pela aplicação sucessiva das sete regras, expostas na seção 2.4.2, no conjunto de triplas que representam a expressão original. À medida que as regras são aplicadas, triplas vão sendo eliminadas reduzindo-se o tamanho do problema original.

2.6.3 Análise, Diagnóstico e Retrocesso em Conflitos

Diferente do processo de dedução em que até hoje as principais ferramentas se baseiam no método BCP proposto por Davis e Loveland (1962), muitas foram as evoluções e propostas de heurísticas para análises, diagnósticos, retrocessos e aprendizados relacionados a conflitos.

Algumas das técnicas propostas na ferramenta Berkimin são frequentemente referenciadas na comunidade SAT (GOLDBERG; NOVIKOV, 2002). Neste solucionador, para o diagnóstico de conflitos, são utilizados grafos que representam as atribuições dadas às variáveis em cada etapa do processo de resolução (MARQUES-SILVA; SAKALLAH, 1999). Esta ferramenta também incorpora uma técnica de aprendizado por conflitos, onde gradativamente novas cláusulas relacionadas a conflitos são adicionadas à expressão inicial. Essas cláusulas são construídas com base em uma análise de literais mais ativos em relação aos conflitos. A limpeza do banco de cláusulas também é feita com base no índice de atividade dos literais, sendo eliminadas as cláusulas com o maior número de literais inativos.

Destaca-se também a técnica de *restart*, que caracteriza-se pelo re-inicio no processo de resolução; devido às novas cláusulas adicionadas ao problema, grandes ramificações da árvore de busca não precisarão mais serem avaliadas.

Melhorias interessantes das técnicas propostas em (GOLDBERG; NOVIKOV, 2002) podem ser encontradas nos trabalhos (AUDEMARD; SIMON, 2009b), (BIERE, 2008) e (RYVCHIN; STRICHMAN, 2008). No algoritmo de Stalmarck o procedimento Dilema equivale aos processos de análise, diagnóstico e retrocesso em conflitos.

3 A FERRAMENTA PUC-SAT

No capítulo anterior foram abordadas as técnicas e mecanismos de funcionamento presentes nos algoritmos de Stalmarck e DPLL. Este capítulo tem por objetivo apresentar a ferramenta PUC-SAT, baseada em ambos conceitos.

3.1 PUC-SAT

Não é difícil observar que uma expressão tautológica possui uma semelhança em relação a uma contradição - enquanto para a primeira tem-se uma tabela verdade que encerra somente a verdade (V), para qualquer combinação de valores das variáveis originais, para a segunda tem-se uma tabela verdade que encerra somente falso (F). Assim sendo o método de Stalmarck também pode ser utilizado para verificar se uma expressão proposicional é uma contradição ou seja NÃO SAT. Basta avaliar a hipótese de que a fórmula original é equivalente à verdade (V).

Com base nesta observação foram criadas duas versões da ferramenta PUC-SAT. A primeira versão (PUC-SAT-01) foi criada com base no tutorial proposto por Sheeran e Stalmarck (1998) e pode ser considerada uma implementação de código aberto para o algoritmo de Stalmarck - com a restrição de que esta versão só se estende até 2-Saturação, abortando a execução para problemas que extrapolam este nível de complexidade. A segunda versão (PUC-SAT-02) possui uma etapa preliminar baseada nos mecanismos propostos por Davis e Loveland (1962) e também possui adaptações relacionadas as sete regras de Stalmarck. A partir dos experimentos efetuados, ambas versões apresentaram bons resultados para instâncias NÃO SAT industriais, sendo a segunda mais eficiente que a primeira. No próximo capítulo serão apresentadas as análises dos resultados encontrados.

Uma importante situação observada durante a avaliação destas duas versões foi a ineficiência do método de Stalmarck para avaliar instâncias que representam contingências. O que é o caso da maioria das intâncias satisfazíveis, pois existem uma ou mais atribuições de valores às variáveis que satisfazem uma expressão, mas também existem atribuições que

não satisfazem a mesma. Com base nesta observação, passou-se a avaliar alternativas para que se tivesse um desempenho mais equilibrado para todo tipo de instância do problema SAT. Resultando na criação de uma terceira versão da ferramenta (PUC-SAT-03).

Um esboço e algumas explicações sobre os mecanismos de funcionamento das três versões do PUC-SAT serão expostos a seguir, considerando: Φ = uma expressão proposicional no formato FNC; Π = um conjunto de triplas equivalente a Φ; "T" = número de triplas; "TR" = número de triplas resolvidas; "v" = uma variável proposicional e **(*)** observações a serem feitas posteriormente.

Algorithm 2 Primeira Etapa PUC-SAT - Baseada em DPLL

```
Primeira Etapa(Φ) (*1)
    Leitura da expressão proposicional no formato FNC
    Enquanto existir "literais puros" ou "cláusulas com tautologia" ou "cláusulas unitárias"
        Remova cláusulas com literais puros
        Remova cláusulas com tautologia
        Execute o BCP
    Fim Enquanto
    Se(Φ = Vazio)
        Resultado = SAT
    Senão
        Resultado = Φ
    Fim Se
Fim Primeira Etapa
```

Em uma primeira etapa **(*1)**, exposta no algoritmo 2 e utilizada nas versões 2 e 3, efetua-se a leitura da instância SAT e executa-se os procedimentos básicos de dedução utilizados pelo algoritmo DPLL, com o intuito de simplificar a expressão original. A estrutura de dados assim como os métodos utilizados nesta etapa foram inspirados no solucionador zChaff (MOSKEWICZ C. F. MADIGAN; MALIK, 2001).

Na segunda etapa, exposta no algoritmo 3, o solucionador utiliza o método de transformação para converter uma expressão FNC em uma expressão no formato de implicação, representando-a com um conjunto de triplas. **(*2)** Para a versão 3 foram realizadas algumas modificações neste método em relação a proposta original, presente em Sheeran e Stalmarck (1998). Tais modificações serão detalhas posteriormente. A seguir o conjunto de triplas é submetido às deduções decorrentes da utilização das sete regras. **(*3)** Para as versões 2 e 3, algumas modificações foram realizadas neste procedimento. Tais adaptações serão detalhas posteriormente. Após a execução das sete regras, diferente das versões 1 e 2 que trabalham com o método **(*4)** dilema, a versão 3 não trabalha com este método mas sim com uma estratégia **(*5)** similar à proposta no algoritmo DPLL para tratamento

Algorithm 3 Segunda Etapa PUC-SAT - Baseada no Método de Stalmarck

```
Segunda Etapa(Φ)
    Função Transformação(Φ) (*2)
        Expressão de Implicação = Π
    Fim Transformação
    Função Resolução(Π)
        Enquanto possível execute Sete Regras (*3)
            Se(contradição)
                Resultado = SAT
            Fim Se
        Fim Sete Regras
        Dilema até 2-Saturação(Π) (*4)
        Se(TR < T)(*5)
            escolha-variável-original(v)
            Resultado = (Resolução( Π ∧ v) ou Resolução(Π ∧ ¬ v))
        Senão
            Resultado = NÃO SAT
        Fim Se
    Fim Resolução
Fim Segunda Etapa
```

de conflitos. Tal estratégia será explicada posteriormente.

A seguir serão detalhadas importantes modificações feitas em alguns métodos presentes nos algoritmos de Stalmarck e DPLL, utilizados nas 3 versões do solucionador PUC-SAT.

3.2 Redução do número de triplas

Conforme detalhado no capítulo anterior, o algoritmo de Stalmarck possui uma etapa preliminar. O processo de transformação da instância SAT, no formato FNC, para um conjunto de triplas - que representam a instância na forma de uma implicação.

Após o processo de transformação, as variáveis presentes na instância original são mantidas no problema e novas variáveis são adicionadas ao mesmo, para representar as implicações. O método de transformação de uma instância no formato FNC para o formato de implicação consiste na aplicação recorrente das regras de transformação.

Analisando-se o processo de transformação proposto por Stalmarck não é difícil perceber que o número de variáveis de implicação adicionadas ao problema cresce consideravelmente, sendo este nas proporções expostas na Tabela 1. Para cada conectivo do tipo $\neg$, $\vee$ e $\wedge$ presente na fórmula original criam-se respectivamente 1, 2 ou 3 novas variáveis

de implicação.

Tabela 1: Variáveis de implicação adicionadas

Variáveis Adicionadas	**Por Conectivo**
2	A $\vee$ B
3	A $\wedge$ B
1	$\neg$ A

Note que o aumento do número de variáveis de implicação está relacionado ao aumento do número de triplas, porque uma variável de implicação representa ou é o resultado de uma tripla. Tal fato pode ser observado a seguir, através das transformações:

Transformação 1

A $\vee$ B $\equiv \neg$ A $\rightarrow$ B $\equiv$ (X1,A,0) (X2, X1,B)

(Adicionadas: X1, X2)

Transformação 2

A $\wedge$ B $\equiv \neg$ (A $\rightarrow \neg$ B) $\equiv$ (X1,B,0) (X2,A, X1) (X3, X2,0)

(Adicionadas: X1, X2, X3)

Transformação 4

$\neg$ A $\equiv$ A $\rightarrow$ 0 $\equiv$ (X1,A,0)

(Adicionada: X1)

Com base no conceito de equivalência lógica, foram feitas **deduções** baseadas nas regras de transformação propostas por Stalmarck, objetivando a redução do número de triplas geradas após o processo de transformação. As seguintes extensões (deduções) foram propostas:

Transformação 1

Dedução 1: A $\vee$ B $\equiv \neg$ A $\rightarrow$ B $\equiv$ (X1,A,0) (X2, X1,B) $\equiv$ **(X1,¬A,B)**

As expressões geradas são equivalentes à expressão original, como pode ser comprovado pelas Tabelas Verdade 2, 3 e 4.

Tabela 2: Tabela verdade - transformação 1 - passo 1

A	**B**	**¬A**	**¬¬A**	**A ∨ B**	**¬¬A ∨ B**	**¬A → B**
0	0	1	0	0	0	0
0	1	1	0	1	1	1
1	0	0	1	1	1	1
1	1	0	1	1	1	1

Tabela 3: Tabela verdade - transformação 1 - passo 2

-	**-**	**X1**	**-**	**X2**	**X2**
A	**B**	**¬A**	**¬X1**	**¬X1 ∨ B**	**X1 → B**
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	1	1	1

Tabela 4: Tabela verdade - transformação 1 - dedução 1

-	**-**	**-**	**-**	**X1**	**X1**
A	**B**	**¬A**	**¬¬A**	**¬¬A ∨ B**	**¬A → B**
0	0	1	0	0	0
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	1	1	1

Transformação 2

Dedução 2: $A \wedge B \equiv \neg (A \rightarrow \neg B) \equiv$ (X1,B,0) (X2,A, X1) (X3, X2,0) $\equiv$ **(X1,A,¬B) (X2, X1,0)**

Dedução 3: (X1,A,¬B) (X2, X1,0) $\equiv$ **(¬X1,A,¬B)**

As expressões geradas são equivalentes à expressão original, como pode ser comprovado pelas Tabelas Verdade 5, 6 , 7 e 8.

Tabela 5: Tabela verdade - transformação 2 - passo 1

A	**B**	**A ∧ B**	**¬A**	**¬B**	**¬A ∨ ¬B**	**A → ¬B**	**¬(A → ¬B)**
0	0	0	1	1	1	1	0
0	1	0	1	0	1	1	0
1	0	0	0	1	1	1	0
1	1	1	0	0	0	0	1

Tabela 6: Tabela verdade - transformação 2 - passo 2

-	**-**	**-**	**X1**	**X2**	**X2**	**X3**	**X3**
A	**B**	**¬A**	**¬B**	**¬A ∨ X1**	**A → X1**	**¬X2 ∨ 0**	**X2 → 0**
0	0	1	1	1	1	0	0
0	1	1	0	1	1	0	0
1	0	0	1	1	1	0	0
1	1	0	0	0	0	1	1

Tabela 7: Tabela verdade - transformação 2 - dedução 2

-	**-**	**-**	**-**	**X1**	**X1**	**X2**	**X2**
A	**B**	**¬A**	**¬B**	**¬A ∨ ¬B**	**A → ¬B**	**¬X1 ∨ 0**	**X1 → 0**
0	0	1	1	1	1	0	0
0	1	1	0	1	1	0	0
1	0	0	1	1	1	0	0
1	1	0	0	0	0	1	1

Tabela 8: Tabela verdade - transformação 2 - dedução 3

-	**-**	**-**	**-**	**X1**	**X1**	**X1**
A	**B**	**¬A**	**¬B**	**¬A ∨ ¬B**	**A → ¬B**	**¬(A → ¬B)**
0	0	1	1	1	1	0
0	1	1	0	1	1	0
1	0	0	1	1	1	0
1	1	0	0	0	0	1

É importante notar que as deduções realizadas são resultantes de uma simplificação decorrente da NÃO aplicação da transformação 4, onde uma proposição ¬A é transformada para A → 0. A aplicação desta transformação resulta em algumas características importantes, como:

- o número de variáveis de implicação gerado após a transformação é maior, na proporção exposta pela Tabela 1.

- o conjunto de triplas gerado após a transformação não possui variáveis negadas (¬). Consequentemente o processo de dedução das triplas, movido pela aplicação das sete regras de Stalmarck, não precisa levar em consideração a possibilidade de existência de variáveis negadas.

- a aplicação da transformação 4 incorre na geração das triplas (X1,A,0), (X1,B,0) e (X3, X2,0), presentes no exemplo. Triplas estas mapeáveis na regra 5 (decisão/dedução): se (R,X,0) então R = ¬X. Assim sendo, observe que a não aplicação da transformação 4 incorrerá em menos deduções a partir da regra 5, quando da primeira execução das sete regras. A esta primeira execução dá-se o nome de Saturação-0.

Para possibilitar as deduções propostas, foram necessárias algumas modificações nas propostas originais de transformação, decisão e dedução feitas em Sheeran e Stalmarck (1998):

- A primeira modificação diz respeito à representação das variáveis, também em sua fase negada, nas triplas. Para isto foram adicionados à estrutura de dados tripla, três indicadores de fase, um para cada variável presente na tripla;

- A segunda foi a alteração do método de transformação para que o mesmo gere as triplas conforme as deduções apresentadas anteriormente;

- E finalmente a re-estruturação do método das sete regras para que este leve em consideração a fase das variáveis presentes nas triplas. As adaptações foram feitas conforme o exemplo, exposto a seguir.

Exemplo de adaptação da regra 1:

Regra 1 (decisão/dedução) : Se (0,X,Y) então X = 1 e Y = 0

Como X e Y podem aparecer negadas, são possíveis também:

- (0,¬X,¬Y) então X = 0 e Y = 1

- (0,¬X,Y) então X = 0 e Y = 0

- (0,X,¬Y) então X = 1 e Y = 1

Observando a Tabela 9 pode-se perceber que as deduções feitas sobre as regras de transformação reduzem o número de variáveis de implicação geradas durante o processo de transformação.

Tabela 9: Comparativo de variáveis de implicação adicionadas

Originalmente	**Deduções 1 e 2**	**Dedução 3**	**Por Conectivo**
2	1	1	A ∨ B
3	2	1	A ∧ B
1	1	0	¬ A

3.3 Análises sobre as sete regras

O capítulo anterior apresenta as sete regras de decisão/dedução propostas por Stalmarck que são responsáveis pelo processo de resolução da expressão representada por um conjunto de triplas. Recordando:

As sete regras de Stalmarck

- Regra 1: (0,X,Y) ⇒ X = 1 e Y = 0
- Regra 2: (R,X,1) ⇒ R = 1
- Regra 3: (R,0,Y) ⇒ R = 1
- Regra 4: (R,1,Y) ⇒ R = Y
- Regra 5: (R,X,0) ⇒ R = ¬X
- Regra 6: (R,R,Y) ⇒ R = 1 e Y = 1
- Regra 7: (R,X,X) ⇒ R = 1

É importante observar que estas regras mapeiam uma relação de causa e consequência. Por exemplo, pela regra 1, cada tripla no formato (R = 0 , X = livre, Y = livre) causa as conseqüências X = 1 e Y = 0. Analisando-se esta situação, algumas deduções podem ser tomadas com base nas sete regras originais e nos conceitos da algebra proposicional.

Objetivando aumentar o número de mapeamentos e consequentemente acelerar o processo de dedução. Ampliando-se as regras 1, 2 e 3:

Regra 1: (0,X,Y) $\Rightarrow$ X = 1 e Y = 0

- Deduções - (tripla de entrada) $\Rightarrow$ consequências:

  (R,1,0) $\Rightarrow$ R = 0

Regra 2: (R,X,1) $\Rightarrow$ R = 1

- Deduções:

  (R,0,1) $\Rightarrow$ R = 1

  (R,1,1) $\Rightarrow$ R = 1

Regra 3: (R,0,Y) $\Rightarrow$ R = 1

- Deduções:

  (R,0,0) $\Rightarrow$ R = 1

  (R,0,1) $\Rightarrow$ R = 1

Outro importante aspecto deve ser avaliado: o impacto causado nas sete regras de decisão/dedução decorrente da possibilidade de representeção da negação $\neg$ nas variáveis presentes nas triplas. Antes de se evidenciar tal impacto, dois conceitos serão necessários para a compreensão do mesmo.

- Decisão: ato de decidir o valor de uma determinada variável. Pela regra 2 tem-se que, ao encontrar uma tripla no formato (R,X,1) pode-se decidir pelo valor de R ser igual a 1 (verdadeiro).
- Propagação: após uma decisão, como a que foi citada, é necessária a propagação da mesma para o restante das triplas. Neste caso substitui-se a variável R por 1 (verdadeiro) e $\neg$R por 0 (falso). Note que uma propagação pode ser de valor, como é o caso gerado pela regra 2, ou pode derivar uma redução da expressão original, consequência da aplicação das regras 4 e 5.

Com base nestes conceitos serão avaliados os cuidados a serem tomados em algumas situações:

- **Inversão de valores na "Regra 1"**. Agora podem ser encontradas as situações (0,X,¬Y),(0,¬X,Y) e (0,¬X,¬Y). Nestes casos as respectivas decisões corretas em relação aos valores de "X" e "Y" são (X = 1 e Y = 1), (X = 0 e Y = 0) e (X = 0 e Y = 1).

- **Re-leitura da "Regra 4"**. A partir das triplas (R,1,¬Y), (¬R,1,Y) e (¬R,1,¬Y), novas decisões devem ser tomadas:

  Possibilidade 1: (R,1,¬Y)

  $R \equiv 1 \rightarrow \neg Y$

  $R \equiv \neg 1 \vee \neg Y$

  $R \equiv 0 \vee \neg Y$

  $R \equiv \neg Y$

  Ações: resolva a tripla e propague R = ¬Y e ¬R = Y.

  Possibilidade 2: (¬R,1,Y)

  $\neg R \equiv \neg(1 \rightarrow Y)$

  $\neg R \equiv \neg(\neg 1 \vee Y)$

  $\neg R \equiv \neg(0 \vee Y)$

  $\neg R \equiv \neg(Y)$

  $\neg R \equiv \neg Y$

  Ações: resolva a tripla e propague R = Y e ¬R = ¬Y.

  Possibilidade 3:(¬R,1,¬Y)

  $\neg R \equiv \neg(1 \rightarrow \neg Y)$

  $\neg R \equiv \neg(\neg 1 \vee \neg Y)$

  $\neg R \equiv \neg(0 \vee \neg Y)$

  $\neg R \equiv \neg(\neg Y)$

  $\neg R \equiv \neg\neg Y$

  $\neg R \equiv Y$

  Ações: resolva a tripla e propague R = ¬Y e ¬R = Y.

O exemplo a seguir demonstra as substituições referentes a possibilidade 3 (¬R,1,¬Y):

(1,¬B,A)(¬B,1,¬A)

Expressão equivalente ao exemplo: $1 \equiv \neg(1 \rightarrow \neg A) \rightarrow A$

Note que a tripla (¬B,1,¬A) equivale a ¬(1 -> ¬A), resolvendo esta expressão tem-se:

- $\neg(\neg 1 \vee \neg A)$
- $\neg(0 \vee \neg A)$
- $\neg(\neg A)$
- A

Observe que a propagação a ser feita é ¬B = A, ou seja a expressão que era representada por "¬B" é equivalente a "A", resultando em (1,A,A).

Expressão equivalente:

- $1 \equiv A \rightarrow A$
- $1 \equiv \neg A \vee A$
- $1 \equiv 1$ (Resolvido)

- **Re-leitura da "Regra 5"**, a partir das novas possibilidades (R,¬X,0), (¬R,X,0) e (¬R,¬X,0).

  Possibilidade 1: (R,¬X,0)

  $R \equiv \neg X \rightarrow 0$

  $R \equiv \neg\neg X \vee 0$

  $R \equiv X \vee 0$

  $R \equiv X$

  Ações: resolva a tripla e propague R = X e ¬R = ¬X.

  Possibilidade 2: (¬R,X,0)

  $\neg R \equiv \neg(X \rightarrow 0)$

  $\neg R \equiv \neg(\neg X \vee 0)$

  $\neg R \equiv \neg(\neg X)$

  $\neg R \equiv \neg\neg X$

  $\neg R \equiv X$

Ações: resolva a tripla e propague R = ¬X e ¬R = X.

Possibilidade 3:(¬R,¬X,0)

¬R ≡ ¬(¬X → 0)

¬R ≡ ¬(¬¬X ∨ 0)

¬R ≡ ¬(X ∨ 0)

¬R ≡ ¬(X)

¬R ≡ ¬X

Ações: resolva a tripla e propague R = X e ¬R = ¬X.

- **Re-leitura da "Regra 6"**. A partir da Tabela 10 pode-se notar que R = 1 e Y = 1 é a única combinação de valores que resolve uma tripla no formato (R,R,Y). Nesta mesma tabela estão destacados em negrito os valores a serem atribuídos a R e Y a partir das novas triplas (R,R,¬Y), (¬R,¬R,Y) e (¬R,¬R,¬Y).

Tabela 10: Re-leitura da regra 6

-	**-**	**Regra 6**	**Possibilidade 1**	**Possibilidade 2**	**Possibilidade 3**
-	**-**	**(R,R,Y)**	**(R,R,¬Y)**	**(¬R,¬R,Y)**	**(¬R,¬R,¬Y)**
R	**Y**	**R ≡ ¬R ∨ Y**	**R ≡ ¬R ∨ ¬Y**	**¬R ≡ ¬¬R ∨ Y**	**¬R ≡ ¬¬R ∨ ¬Y**
0	0	0 ≡ 1 ∨ 0	0 ≡ 1 ∨ 1	1 ≡ 0 ∨ 0	**1 ≡ 0 ∨ 1**
0	1	0 ≡ 1 ∨ 1	0 ≡ 1 ∨ 0	**1 ≡ 0 ∨ 1**	1 ≡ 0 ∨ 0
1	0	1 ≡ 0 ∨ 0	**1 ≡ 0 ∨ 1**	0 ≡ 1 ∨ 0	0 ≡ 1 ∨ 1
1	1	**1 ≡ 0 ∨ 1**	1 ≡ 0 ∨ 0	0 ≡ 1 ∨ 1	0 ≡ 1 ∨ 0

Para finalizar é necessária uma consideração importante sobre as regras que alteram o valor da variável "R", como é o caso das regras 2, 3, 4, 5, 6 e 7. Esta variável representa o resultado de uma tripla, caso a variável esteja em fase negada "¬R" o valor a ser atribuído **NÃO** deve ser invertido. Por exemplo, caso apareça uma tripla do tipo (¬R,X,1) tem-se um mapeamento na regra 2 que diz para realizar a decisão "R = 1". A decisão a ser tomada é realmente esta e não inverter o valor de "R" para "R = 0", em busca do resultado (1,X,1). Interprete uma negação na variável "R" como uma ação a ser tomada após a execução das regras, ou seja, entenda a tripla (¬R,X,1) da seguinte forma ¬[(R,X,1)]⇒(regra 2)¬[(1,X,1)]⇒¬[1]⇒0.

3.4 O dilema recursivo

Para analisar a ineficiência do algoritmo de Stalmarck em lidar com instâncias satifatíveis (contingências), tome como exemplo a expressão (A) ∧ (B). Note que esta instância não é uma tautologia e também não é uma contradição, ou seja é uma contingência. Analisando a resolução Figuras 1 e 2 pode-se notar a necessidade do algoritmo utilizar o método dilema até a N-Saturação.

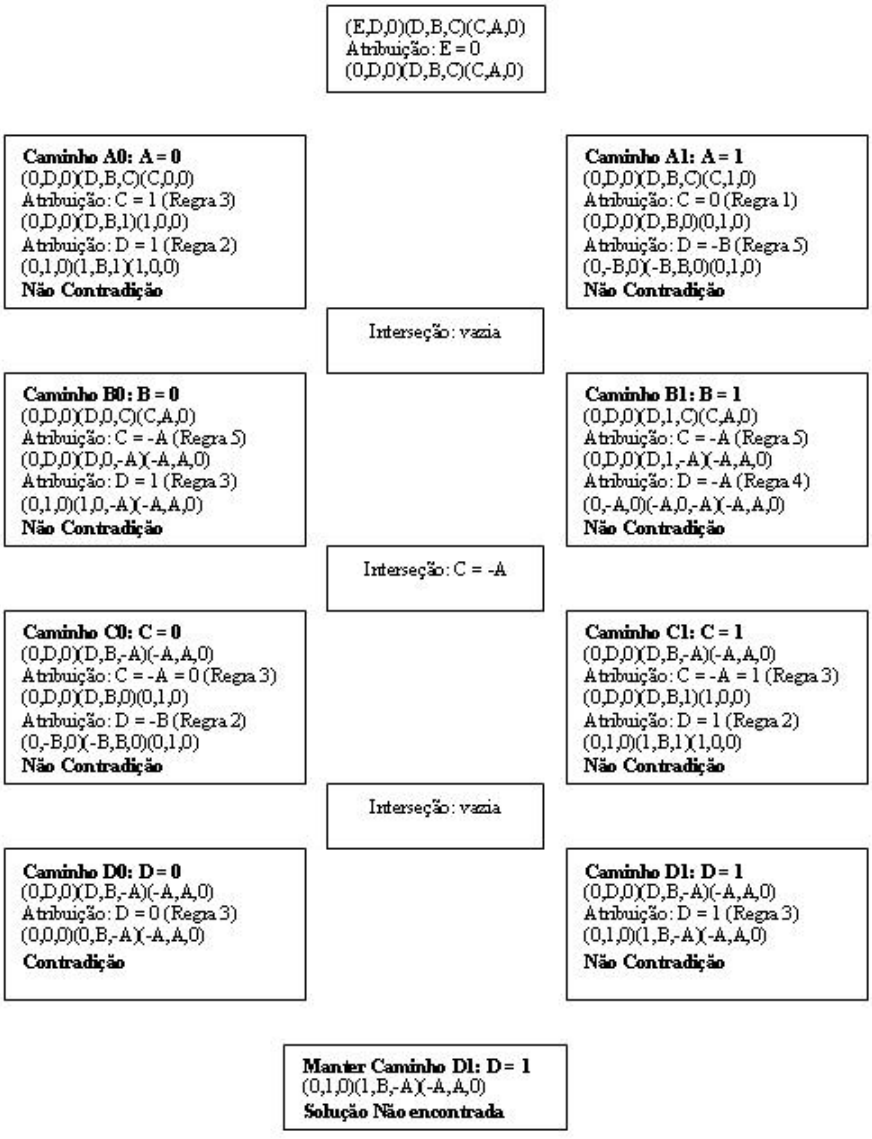

Figura 1: Exemplo de resolução (A) ∧ (B) em 1-Saturação

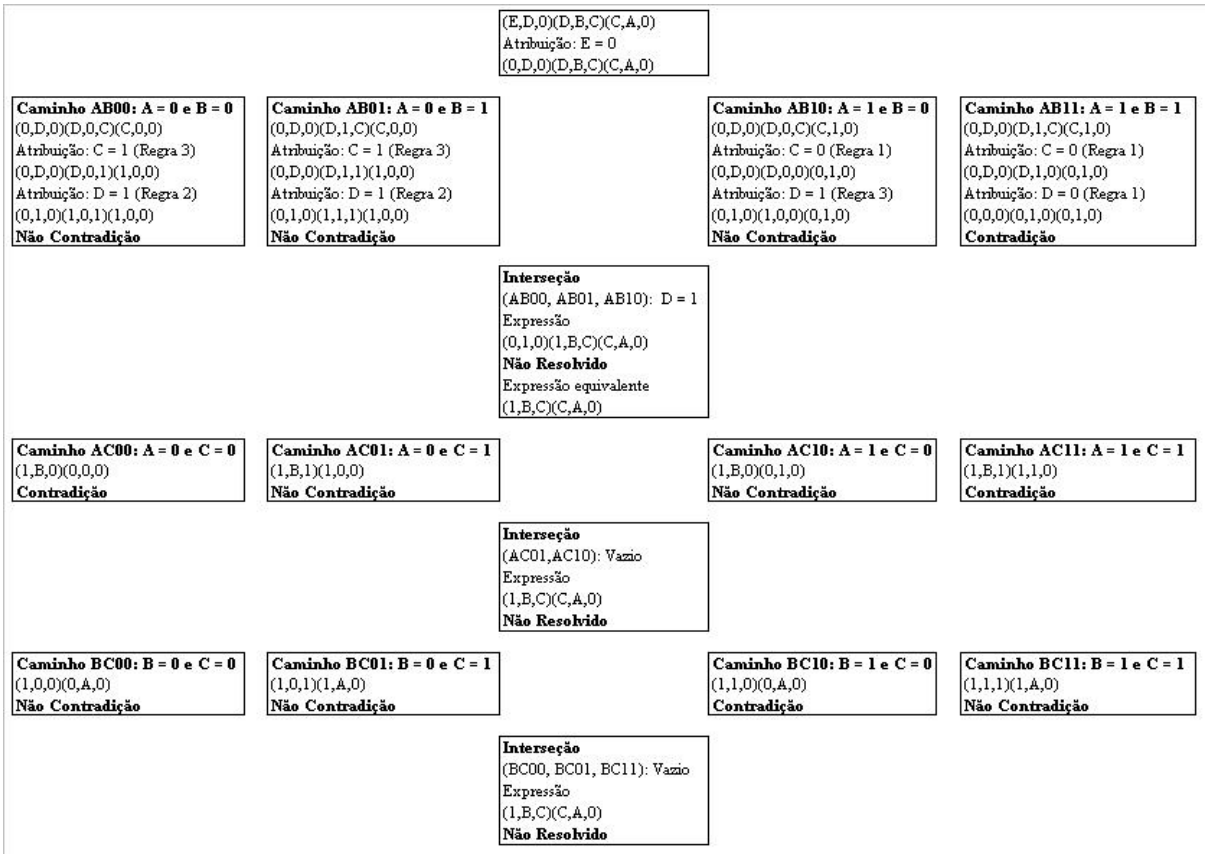

Figura 2: Exemplo de resolução (A) $\wedge$ (B) em 2-Saturação

Tal situação acontece, em instâncias que são contingências, tanto para o método de avaliação de tautologia quanto para o de avaliação de contradição. Este fato leva a avaliação de novas perspectivas para o tratamento deste tipo de instância. Uma solução, implementada no PUC-SAT-03, é a substituição do método dilema por um caminhamento em profundidade no universo de soluções, através de um procedimento recursivo, assim como o proposto em Davis e Loveland (1962).

Através desta solução, é possível avaliar o comportamento do mecanismo de dedução baseado nas sete regras de Stalmarck, aliado a estratégia de busca com retrocesso do algoritmo DPLL. Com base nestes conceitos foi implementada a solução exposta no algoritmo 4, que é parte da segunda etapa de resolução da ferramenta PUC-SAT-03.

Algorithm 4 Função Sete Regras com Dilema Recursivo

```
Função SeteRegras(Variável, ValorDaVariável)
  Variável = ValorDaVariavel
  Execute as sete regras enquanto possível
  Se (Contradicao)
      Resultado = SAT
  Fim Execute
  EscolhaUmaVariávelOriginal(Variável)
  Se((SeteRegras(Variável,1)=NÃO SAT)ou(SeteRegras(Variável,0)=NÃO SAT))
      Resultado = SAT
  Senão
      Resultado = NÃO SAT
Fim SeteRegras
```

Tal solução possui algumas particularidades interessantes. Uma delas diz respeito ao método de escolha de variáveis, que serão levadas à divisão (tentar valor 0, depois tentar valor 1). Tais variáveis têm que ser variáveis originais do problema, não podendo ser variáveis de implicação. Isto pelo fato das variáveis de implicação serem apenas representações do conectivo de implicação "$\rightarrow$". Por exemplo, convertendo uma expressão $\neg A \vee B$ para o formato de triplas, ter-se-ia a tripla (C,A,B). Sendo a variável de implicação C a representação ou o resultado do conectivo "$\rightarrow$", na expressão $C \equiv A \rightarrow B$. Assim sendo pode-se determinar o valor de C com base nos valores de A e B, mas não o contrário. Quando se tenta atribuir valores arbitrários a C durante o processo de dedução situações inconsistentes são geradas, impossibilitando o correto funcionamento do mecanismo de resolução.

Outro aspecto importante no método proposto é que, diferente do algoritmo de Stalmarck, este pode parar quando uma contradição for encontrada em somente um dos caminhos pesquisados - não sendo necessário encontrar contradição nos dois caminhos possíveis. Quando uma contradição é encontrada tem-se a solução para o problema SAT dada pelos valores atuais das variáveis originais, com pode ser visto na Figura 3.

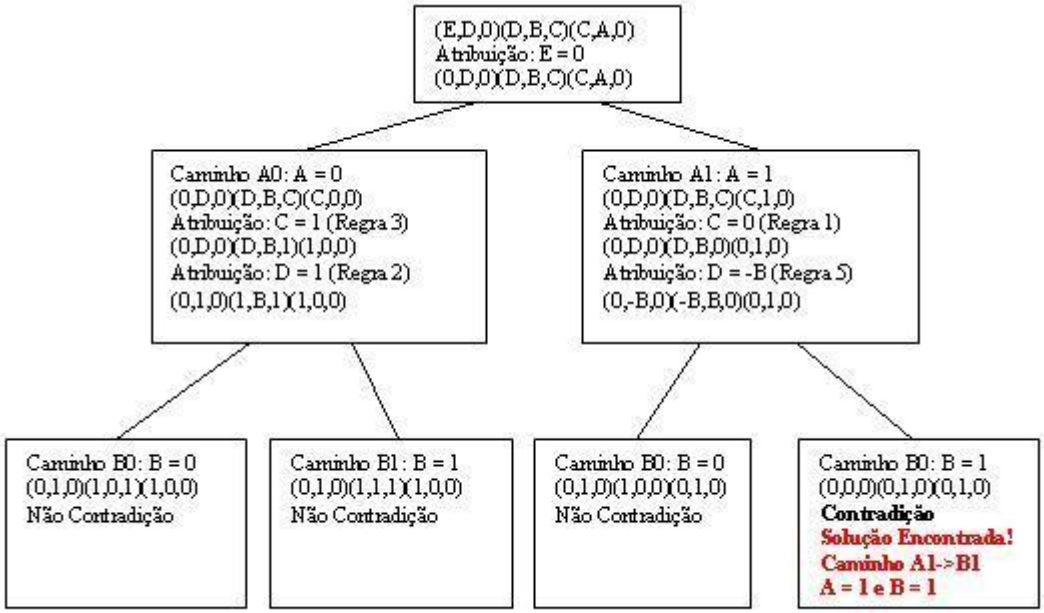

Figura 3: Exemplo de resolução (A) $\wedge$ (B) pelo Puc-SAT

Analisando esta solução pode-se ter a impressão de que todos os 2^n possíveis caminhos serão investigados. Mas tal fato não é verdade levando-se em consideração que as "sete regras" remetem a decisões quanto a valores de variáveis tanto de implicação quanto originais. Quando uma decisão quanto ao valor de uma variável original é tomada, não será mais necessário investigar as possibilidades relacionadas ao valor oposto ao escolhido. O universo de busca é reduzido a cada decisão tomada no método "sete regras". Um exemplo detalhado mostrando a execução dos procedimentos expostos neste capítulo pode ser visto na Figura 4.

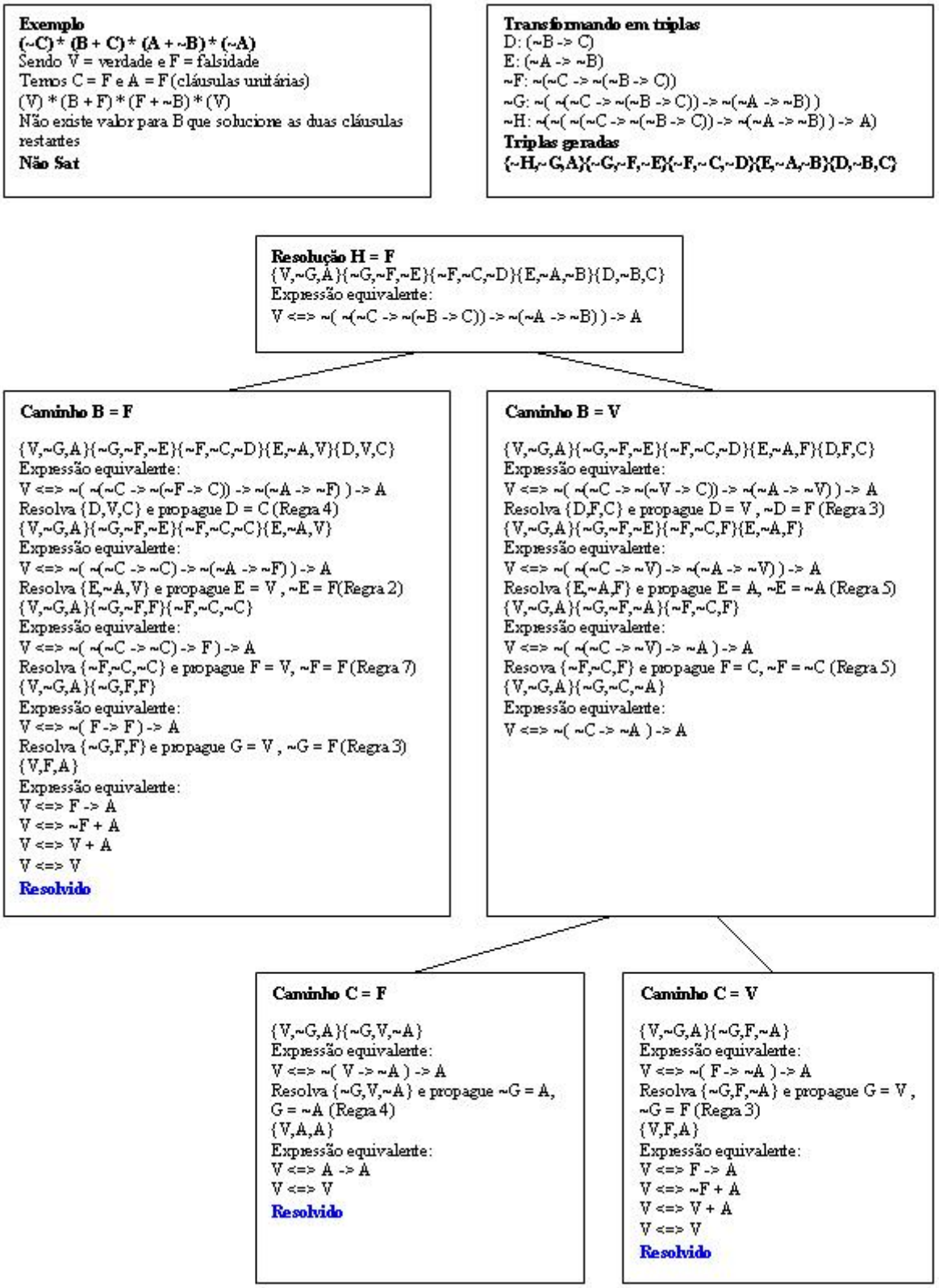

Figura 4: Exemplo de resolução $(\neg C) \wedge (B \vee C) \wedge (A \vee \neg B) \wedge (\neg A)$ pelo Puc-SAT

4 RESULTADOS

Neste capítulo são apresentados os resultados alcançados pelas três versões da ferramenta PUC-SAT. Analisando-se também o comportamento da mesma em relação aos solucionadores ZChaff (ZCHAFF-HOMEPAGE, 2010) e RSat (RSAT-HOMEPAGE, 2010).

4.1 Instâncias

Para as análises foram utilizadas cerca de duzentas instâncias, retiradas das competições SAT. Todas as tabelas que serão expostas neste capítulo são amostras retiradas do universo de testes e possuem tempos de execução expostos em segundos, sendo o limite para *time out* de 3600s.

A máquina utilizada para os testes foi um *Pentium Dual Core* com 1GB de memória. Devido ao grande volume de variáveis geradas pelo método de transformação de uma instância FNC para o formato de implicação, os testes realizados apresentam uma restrição: não foram utilizadas instâncias com mais de 30.000 variáveis. Instâncias acima deste limite levaram a máquina, utilizada nos testes, a altas taxas de utilização de memória secundária, comprometendo os resultados. Descrições das instâncias utilizadas podem ser observadas a seguir:

Instâncias Industriais

- *bmc*.cnf => problemas *BMC - bounded model checking.*
- eq*.cnf => problemas de verificação de equivalência em hardware.
- phyhala*.cnf => problemas construídos. Forma de construção: *A permuted SAT Competition formula with seed=888363780.*
- *pipe*.cnf => fórmulas geradas na verificação formal de microprocessadores superescalares. Fonte: Miroslav Velev's SAT *Benchmarks.*

- ndhf-xits-*.cnf / rbcl-xits-*.cnf / rpoc-xits-*.cnf => Bio-Instâncias submetidas à competição SAT de 2009.

Outras Instâncias

- aim*.cnf => instâncias artificiais geradas a partir de um gerador. Fonte: Kazuo Iwama, Eiji Miyano (miyano@cscu.kyushu-u.ac.jp) e Yuichi Asahiro
- *bit*.cnf => problemas diversos relacionados a movimentação de bits.
- dubois*.cnf => instâncias do gerador gensathard.c. Fonte: Olivier Dubois (dubois@laforia.ibp.fr)
- jnh*.cnf => problemas randômicos com gerador *Hard* (sem cláusulas únitárias, densidade duro, etc). Fonte: John Hooker (jh38 + @ andrew.cmu.edu)
- hole*.cnf => *Pigeon Hole Problem* -> colocação de n pombos em n +1 buracos sem colocar dois pombos no mesmo buraco. Fonte: John Hooker (jh38 + @ andrew.cmu.edu)
- par*.cnf => instâncias derivadas do problema de aprendizagem da função de paridade. Parxx-y denota um problema de paridade em xx bits, sendo y o número da instância. Parxx-y-c denota um caso idêntico a Parxx-y, exceto pelo fato da instância estar simplificada (para criar um problema equivalente).
- mod2-rand*.cnf / easy*.cnf / unif*.cnf / *blocks*.cnf / ais*.* => instâncias randômicas no formato DIMACS.
- ssa*.cnf / bf*.cnf => fórmulas Nemesis 6CNF em conformidade com o formato CNF DIMACS. Fonte: Allen Van Gelder (avg@cs.ucsd.edu) e Yumi Tsuji (tsuji@cse.ucsc.edu)
- syn*.cnf / puz*.cnf / gra*.cnf => instâncias utilizadas no trabalho de Harrison (1996).

4.2 PUC-SAT-01 x PUC-SAT-02

Conforme o exposto no capítulo anterior a versão PUC-SAT-01 (V-01) foi criada com o objetivo de reproduzir o método descrito no tutorial elaborado por Sheeran e Stalmarck

(1998). Lembrando que esta versão possui duas diferenças em relação ao método original, só executa até 2-Saturação e verifica contradição, não tautologia.

Na versão PUC-SAT-02 (V-02) foi incluída uma etapa preliminar onde são executados procedimentos de decisão e dedução propostos por Davis e Loveland (1962) e também extensões no método sete regras explicadas na seção 3.3.

Com base em testes efetuados com instâncias NÃO satisfazíveis diversas, verificou-se que a estratégia proposta na segunda versão foi em média 25% superior em relação à primeira, em termos de tempo de execução. O que pode ser observado na amostra exposta na Tabela 11.

Tabela 11: PUC-SAT-01 x PUC-SAT-02 (Instâncias NÃO-SAT)

Instância	**Variáveis**	**Cláusulas**	**V-01**	**V-02**
syn051-1.cnf	3	6	0,015	0
gra001-1.cnf	5	12	0,031	0
puz030-2.cnf	10	63	0,609	0,228
quinn.cnf	16	18	0,078	0
hole6.cnf	42	133	2,562	1,924
par8-1-c.cnf	64	254	12,781	9,121
dubois27.cnf	81	216	5,687	4,348
aim-100-1-6-no-3.cnf	100	160	3,297	2,749
jnh211.cnf	100	800	283,063	210,297
ssa0432-001.cnf	435	1027	5,469	4,709
eq.atree.braun.8.unsat.cnf	684	2300	0,156	0,117
2pipe.cnf	892	6695	0,453	0,418
rbcl-xits-06-UNSAT.cnf	980	47620	1457,76	1392,61
bf0432-007.cnf	1040	3668	55,61	45,871
rpoc-xits-08-UNSAT.cnf	1278	74789	2187,13	2099,875
ssa2670-128.cnf	1359	3321	55,438	42,785
eq.atree.braun.12.unsat.cnf	1694	5726	0,406	0,304
ndhf-xits-09-UNSAT.cnf	1910	167931	2679,08	2579,763
ndhf-xits-10-UNSAT.cnf	2112	191788	2898,76	2816,872
3pipe-1-ooo.cnf	2223	26561	2,25	2,111
3pipe-2-ooo.cnf	2400	29981	2,625	2,579
ndhf-xits-13-UNSAT.cnf	2730	271285	Time Out	Time Out
4pipe-1-ooo.cnf	4647	74554	9,906	9,176
4pipe.cnf	5237	80213	13,969	13,696
pyhala-braun-unsat-30-4-01.shuffled.cnf	5428	17845	63,078	47,834
5pipe-1-ooo.cnf	8441	187545	51,593	48,694
pyhala-braun-unsat-40-4-01.shuffled.cnf	9638	31795	244,922	210,124
6pipe.cnf	15800	394739	114,14	107,045
6pipe-6-ooo	17064	545612	200,67	171,002
7pipe	23910	751118	298,13	261,59

4.3 PUC-SAT-01 x PUC-SAT-03

A versão PUC-SAT-03 (V-03) foi desenvolvida devido a ineficiência do método de Stalmarck em trabalhar com instâncias satisfazíveis. Nesta versão foram incluídas a política de redução de triplas (seção 3.2) e o dilema recursivo (seção 3.4).

Com base em testes efetuados com instâncias NÃO SAT e SAT variadas, verificou-se que a estratégia exposta na terceira versão apresentou desempenho pior ao das duas primeiras, para instâncias não satisfazíveis. Em contrapartida observou-se evolução para instâncias satisfazíveis. Mas é importante ressaltar que a V-03 se limita a resolver apenas instâncias pequenas do problema SAT e com desempenho inferior a ferramentas do estado da arte.

Na Tabela 12 pode ser observada uma amostra que comprova a inferioridade da V-03 em relação a V-01 para instâncias NÃO SAT. Esta tabela também mostra o resultado da aplicação do método de redução de triplas, implementado em V-03.

Tabela 12: PUC-SAT-01 x PUC-SAT-03 (Instâncias NÃO SAT)

Instância	**Var.**	**Clau.**	**Triplas-01**	**V-01**	**Triplas-03**	**V-03**
syn051-1.cnf	3	6	33	0,015	11	0,431
gra001-1.cnf	5	12	89	0,031	31	0,827
puz030-2.cnf	10	63	603	0,609	213	2,323
quinn.cnf	16	18	101	0,078	35	1,865
hole6.cnf	42	133	970	2,562	293	598,034
par8-1-c.cnf	64	254	2096	12,781	731	986,372
dubois27.cnf	81	216	1833	5,687	647	886,923
aim-100-1-6-no-3.cnf	100	160	1362	3,297	479	563,923
jnh211.cnf	100	800	10514	283,063	3914	Time Out
ssa0432-001.cnf	435	1027	6954	5,469	2363	3176,821
eq.atree.braun.8.unsat.cnf	684	2300	17388	0,156	5991	1760,344
2pipe.cnf	892	6695	55376	0,453	7827	Time Out
rbcl-xits-06-UNSAT.cnf	980	47620	428254	1457,76	141220	Time Out
bf0432-007.cnf	1040	3668	28641	55,61	9573	Time Out
rpoc-xits-08-UNSAT.cnf	1278	74789	678785	2187,13	222345	Time Out
ssa2670-128.cnf	1359	3321	22460	55,438	7724	Time Out
eq.atree.braun.12.unsat.cnf	1694	5726	43167	0,406	14873	Time Out
ndhf-xits-09-UNSAT.cnf	1910	167931	1524927	2679,08	500697	Time Out
ndhf-xits-10-UNSAT.cnf	2112	191788	1741170	2898,76	572016	Time Out
3pipe-1-ooo.cnf	2223	26561	227020	2,25	76152	Time Out
3pipe-2-ooo.cnf	2400	29981	256631	2,625	86080	Time Out
ndhf-xits-13-UNSAT.cnf	2730	271285	2461221	Time Out	809745	Time Out
4pipe-1-ooo.cnf	4647	74554	645427	9,906	216263	Time Out
4pipe.cnf	5237	80213	691532	13,969	231844	Time Out
pyhala-braun-unsat-30-4-01.shuffled.cnf	5428	17845	133207	63,078	46154	Time Out
5pipe-1-ooo.cnf	8441	187545	1639937	51,593	548816	Time Out
pyhala-braun-unsat-40-4-01.shuffled.cnf	9638	31795	237639	244,922	82344	Time Out
6pipe.cnf	15800	394739	3458132	114,14	1157224	Time Out
6pipe-6-ooo	17064	545612	4889252	200,67	1608427	Time Out
7pipe	23910	751118	6689250	298,13	2211467	Time Out

A Tabela 13 mostra os tempos de resolução da V-03, para algumas pequenas instâncias satisfazíveis. Em contrapartida é possível observar que a V-01, não consegue resolver as mesmas instâncias até a 2-Saturação.

Tabela 13: PUC-SAT-01 x PUC-SAT-03 (Instâncias SAT)

Instância	**Var.**	**Clau.**	**V-01**	**V-03**
easy0.cnf	6	6	> 2-Sat	0,213
easy4.cnf	8	8	> 2-Sat	0,289
easy8.cnf	12	12	> 2-Sat	0,89
4blocks.cnf	41	24758	> 2-Sat	988,035
ais10.cnf	61	581	> 2-Sat	878,012
par8-1-c.cnf	64	254	> 2-Sat	242,13
par8-4-c.cnf	67	266	> 2-Sat	276,81
par8-2-c.cnf	68	270	> 2-Sat	209,11
par8-5-c.cnf	75	298	> 2-Sat	212,99
aim-100-3-4-yes1-1.cnf	100	340	> 2-Sat	1132,99
aim-100-6-0-yes1-3.cnf	100	600	> 2-Sat	1918,056
ais12.cnf	113	1520	> 2-Sat	2967,88
2bitcomp-5.cnf	125	310	> 2-Sat	945,023
ais6.cnf	181	3151	> 2-Sat	Time Out
mod2-rand3bip-sat-210-1.sat05-2158.reshuffled-07.cnf	210	840	> 2-Sat	886,923
mod2-rand3bip-sat-220-1.sat05-2173.reshuffled-07.cnf	220	880	> 2-Sat	989,231
mod2-rand3bip-sat-230-2.sat05-2189.reshuffled-07.cnf	230	920	> 2-Sat	1100,67
mod2-rand3bip-sat-240-3.sat05-2205.reshuffled-07.cnf	240	960	> 2-Sat	1232,11
2bitmax-6.cnf	252	766	> 2-Sat	1342,772
ais8.cnf	265	5666	> 2-Sat	Time Out
3blocks.cnf	283	9690	> 2-Sat	Time Out
par16-1-c.cnf	317	1264	> 2-Sat	1134,191
par16-2-c.cnf	349	1392	> 2-Sat	1087,12
unif-k3-r4.26-v360-c1533-S1217224084-10.SAT.shuffled.cnf	360	1533	> 2-Sat	2018,923
unif-k3-r4.26-v360-c1533-S1422217194-16.SAT.shuffled.cnf	360	1533	> 2-Sat	1768,045
unif-k3-r4.26-v360-c1533-S2109173872-19.SAT.shuffled.cnf	360	1533	> 2-Sat	1871,013
2bitadd-11.cnf	649	1520	> 2-Sat	2332,89
2bitadd-12.cnf	708	1702	> 2-Sat	2561,012
4blocksb.cnf	758	47820	> 2-Sat	Time Out

4.4 PUC-SAT-02 x ZChaff x RSat

Conforme o destacado por Sheeran e Stalmarck (1998), o método de Stalmarck apresenta resultados destacados para problemas industriais. Assim sendo foi feita uma avaliação do desempenho da ferramenta PUC-SAT-02, nesta classe de problemas, em relação a dois solucionadores baseados em DPLL.

O primeiro solucionador escolhido foi o ZChaff (ZCHAFF-HOMEPAGE, 2010), pelo fato da primeira etapa de resolução do PUC-SAT ser baseada em seu código. Já o solucionador RSat (RSAT-HOMEPAGE, 2010) foi escolhido por ser eficiente para instâncias industriais NÃO SAT, sendo o campeão da competição SAT nesta categoria em 2007.

Como já era esperado a solução baseada em Stalmarck não teve bom desempenho para instâncias satisfazíveis. A Tabela 14 mostra os resultados da execução dos três solucionadores para problemas industriais NÃO SAT de diferentes naturezas. Como pode ser observado nesta tabela, o solucionador PUC-SAT-02 apresentou o seguinte resultado, em termos de tempo de execução:

- Para as instâncias pyhala*.cnf foi em média 45% superior ao ZChaff e 80% superior em relação ao RSat.

- Para as instâncias eq*.cnf foi significativamente mais rápido que os outros dois solucionadores, resolvendo os problemas centenas e em alguns casos milhares de vezes mais rápido.

- Para as instâncias *xits*.cnf obteve desempenho superior, pois resolveu a maioria dos problemas dentro dos 3600s enquanto as outras ferramentas não conseguiram resolver as mesmas instâncias no prazo. Destacando que para a instância rbcl-xits-06-UNSAT.cnf o solucioandor RSat obteve desempenho superior e em alguns casos as três ferramentas abortaram por *time out*.

- Para as instâncias *pipe*.cnf foi em média 5% inferior ao ZChaff e 28% superior ao RSat. Destacando que para instâncias com maior volume de variáveis o PUC-SAT-02 foi superior também ao ZChaff.

- Para as instâncias *bmc*.cnf o PUC-SAT-02 não apresentou bom desempenho, só conseguindo resolver a instância hoons-vbmc-lucky7.cnf com desempenho próximo aos dos outros solucionadores.

Observando as intâncias relativas a verificação de equivalência em hardware (eq*.cnf), onde o solucionador PUC-SAT-02 obteve desempenho destacado, pode-se notar uma interessante característica: a maioria das cláusulas presentes nestas fórmulas são de tamanho igual ou inferior a 3 literais, incorrendo na geração de um número pequeno de triplas para se representar as cláusulas e consequentemente a instância.

Por este motivo intui-se que a velocidade da ferramenta esta diretamente relacionada ao número de triplas necessário para se representar a instância, obtendo-se a solução mais rápido para problemas representados por um conjunto de cláusulas pequenas.

Tabela 14: PUC-SAT-02 x ZChaff x RSat (Instâncias Industriais NÃO-SAT)

Instância	**Var.**	**Clau.**	**V-02**	**ZChaff**	**RSat**
pyhala-braun-unsat-30-4-01.shuffled.cnf	5428	17845	47,834	74,771	86,174
pyhala-braun-unsat-30-4-02.shuffled.cnf	5428	17845	41,823	92,31	59,894
pyhala-braun-unsat-30-4-03.shuffled.cnf	5428	17845	1,236	68,912	61,102
pyhala-braun-unsat-30-4-04.shuffled.cnf	5428	17845	16,835	77,111	66,72
pyhala-braun-unsat-35-4-01.shuffled.cnf	7383	24320	324,363	260,008	818,648
pyhala-braun-unsat-35-4-02.shuffled.cnf	7383	24320	83,715	167,067	745,332
pyhala-braun-unsat-35-4-03.shuffled.cnf	7383	24320	198,187	244,009	790,554
pyhala-braun-unsat-35-4-04.shuffled.cnf	7383	24320	35,183	142,23	1125,864
pyhala-braun-unsat-40-4-01.shuffled.cnf	9638	31795	210,124	274,51	Time Out
pyhala-braun-unsat-40-4-02.shuffled.cnf	9638	31795	270,592	282,081	Time Out
pyhala-braun-unsat-40-4-03.shuffled.cnf	9638	31795	33,921	224,045	Time Out
pyhala-braun-unsat-40-4-04.shuffled.cnf	9638	31795	148,023	225,171	Time Out
eq.atree.braun.8.unsat.cnf	684	2300	0,117	9,27	133,215
eq.atree.braun.9.unsat.cnf	892	3006	0,162	72,94	505,156
eq.atree.braun.10.unsat.cnf	1111	3756	0,212	299,17	Time Out
eq.atree.braun.11.unsat.cnf	1400	4732	0,234	1918,11	Time Out
eq.atree.braun.12.unsat.cnf	1694	5726	0,304	Time Out	Time Out
eq.atree.braun.13.unsat.cnf	2010	6802	0,412	Time Out	Time Out
ndhf-xits-09-UNSAT.cnf	1910	167931	2579,763	Time Out	Time Out
ndhf-xits-10-UNSAT.cnf	2112	191788	2816,872	Time Out	Time Out
ndhf-xits-11-UNSAT.cnf	2316	216962	Time Out	Time Out	Time Out
ndhf-xits-12-UNSAT.cnf	2522	243459	Time Out	Time Out	Time Out
ndhf-xits-13-UNSAT.cnf	2730	271285	Time Out	Time Out	Time Out
ndhf-xits-14-UNSAT.cnf	2940	300446	Time Out	Time Out	Time Out
rbcl-xits-06-UNSAT.cnf	980	47620	1392,61	Time Out	80,515
rbcl-xits-07-UNSAT.cnf	1128	57446	1501,97	Time Out	2777,937
rbcl-xits-08-UNSAT.cnf	1278	68055	1623,239	Time Out	Time Out
rpoc-xits-07-UNSAT.cnf	1128	63345	1992,971	Time Out	Time Out
rpoc-xits-08-UNSAT.cnf	1278	74789	2099,875	Time Out	Time Out
rpoc-xits-09-UNSAT.cnf	1430	87044	2402,981	Time Out	Time Out
2pipe.cnf	892	6695	0,418	0,171	0,312
2pipe-1-ooo.cnf	834	7026	0,384	0,167	0,203
2pipe-2-ooo.cnf	925	8213	0,518	0,189	0,156
3pipe.cnf	2468	27533	2,771	2,221	3,14
3pipe-1-ooo.cnf	2223	26561	2,111	1,981	3,64
3pipe-2-ooo.cnf	2400	29981	2,579	2,911	5,828
3pipe-3-ooo.cnf	2577	33270	2,981	3,115	4,906
4pipe.cnf	5237	80213	13,696	16,297	33,653
4pipe-1-ooo.cnf	4647	74554	9,176	16,311	15,093
4pipe-2-ooo.cnf	4941	82207	10,872	35,299	24,261
4pipe-3-ooo.cnf	5233	89473	12,054	20,186	40,156
4pipe-4-ooo.cnf	5525	96480	13,132	22,11	41,203
5pipe.cnf	9471	195452	62,013	32,656	72,578
5pipe-1-ooo.cnf	8441	187545	48,694	60,008	57,812
5pipe-2-ooo.cnf	8851	201796	51,98	55,833	94,593
5pipe-3-ooo.cnf	9267	215440	53,785	57,255	89,765
5pipe-4-ooo.cnf	9764	221405	54,222	113,722	225,776
5pipe-5-ooo.cnf	10113	240892	62,605	66,733	83,406
6pipe.cnf	15800	394739	107,045	172,881	1194,969
6pipe-6-ooo	17064	545612	171,002	305,551	1560,226
7pipe	23910	751118	261,59	611,879	Time Out
hoons-vbmc-lucky7.cnf	8503	25116	31,216	63,54	44,36
cmu-bmc-longmult15.cnf	7807	24351	936,11	268,09	146,19
cmu-bmc-barrel6	2306	8931	Time Out	45,33	5,97

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho descreve a construção de uma ferramenta de código aberto, PUC-SAT. Tal ferramenta tem por objetivo a resolução de instâncias do problema da satisfabilidade, utilizando a aplicação de conceitos dos algoritmos de Stalmarck e DPLL.

Conforme exposto neste trabalho, foram desenvolvidas três versões para a ferramenta PUC-SAT. Com destaque para o PUC-SAT-02 que apresentou bons resultados para instâncias industriais não satisfazíveis. Esta versão possui uma etapa inicial baseada nos métodos de "decisão" e "dedução" presentes no algoritmo DPLL. E em uma segunda etapa o solucionador trabalha com métodos de "transformação", "decisão" e "dedução" baseados no algoritmo de Stalmarck.

Alterações foram propostas em relação a alguns dos métodos originais apresentados por Sheeran e Stalmarck (1998). Tais alterações foram descritas ao longo do trabalho, sendo o seu comportamento analisado.

Com base nas análises realizadas é possível constatar que foi alcançado o objetivo de se reproduzir o trabalho de Stalmarck e comprovar o bom desempenho do mesmo para instâncias que representam tautologias ou contradições. Mostrou-se também que tal proposta, com algumas adaptações, aliadas a estratégias baseadas em DPLL, apresenta bons resultados para instâncias de mesma natureza. As tentativas realizadas no PUC-SAT-03 não comprovaram a possibilidade de se aplicar de forma eficiente o método de Stalmarck combinado a técnicas baseadas em DPLL para instâncias satisfazíveis. Mas não se pode descartar a possibilidade deste objetivo ser alcançado em trabalhos futuros.

Como trabalho futuro almeja-se a inserção de uma heurística de decisão na segunda etapa da ferramenta PUC-SAT-02. Uma ideía inicial seria a escolha das triplas a serem levadas à decisão com base em variáveis mais presentes em toda a instância. Com esta estratégia intui-se que mais triplas sejam resolvidas a cada decisão, tornando o processo de resolução mais rápido.

Outro trabalho interessante a ser realizado tanto na versão 2 quanto na 3 seria o re-

finamento dos procedimentos de decisão, dedução e tratamento de conflitos, baseados em DPLL, conjugando-os com estratégias eficientes para ferramentas desta natureza. Como por exemplo algumas das técnicas descritas na seção 2.6: a utilização de grafos de implicação para a análise de conflitos, o procedimento de *re-start* aliado a inclusão de cláusulas construídas a partir de conflitos, a escolha da variável a ser levada à decisão com base na sua atividade em relação a conflitos, dentre outras.

REFERÊNCIAS

ABE, J. M. *Introdução à lógica para ciência da computação*. [S.l.]: Arte e Ciência, 2002.

AUDEMARD, G.; SIMON, L. Glucose: a solver that predicts learnt clauses quality. *In proceedings of SAT*, 2009.

AUDEMARD, G.; SIMON, L. Predicting learnt clauses quality in modern sat solvers. *In proceedings of IJCAI*, 2009.

BETH, E. W. Semantic entailment and formal derivability. *Mededelingen der Kon. Nederlandse Akademie van Wetenschappen*, v. 18, p. 309–342, 1955.

BIERE, A. Adaptive restart strategies for conflict driven sat solvers. *In proceedings of SAT*, p. 28–33, 2008.

BIERRE ALESSANDRO CIMATTI, E. M. C. A.; ZHU, Y. Sumbolic model checking without bdds. *In Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, 1999.

BUCH, A.; HILLENBRAND, T. *WALDMEISTER: Development of a High Performance Completion-Based Theorem Prover*. [S.l.], 1996.

BURO, M.; BUNING, H. K. Report on a sat competition. *Technical report*, 1992.

CIMATTI E. M. CLARKE, E. G. F. G. M. P. M. R. R. S. A.; TACCHELLA, A. Nusmv 2: An opensource tool for symbolic model checking. In: *In Proceeding of International Conference on Computer-Aided Verification (CAV 2002)*. [s.n.], 2002. p. 27–31. Disponível em: <http://nusmv.irst.itc.it/>.

CLARKE, E. A. E. E. M.; SISTLA, A. P. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, n. 8, p. 244, 1986.

CLARKE, E. M.; EMERSON, E. A. Design and synthesis of synchronization skeletons using branching-time temporal logic. *Logic of Programs*, p. 52–71, 1981.

COMPETITION, S. *The international SAT Competitions*. 2009. Último Acesso: 29/05/2010. Disponível em: <http://www.satcompetition.org/>.

COOK, S. A. The complexity of theorem-proving procedures. In: *Proceedings of the Third IEEE Symposium on the Foundations of Computer Science*. [s.n.], 1971. p. 151–158. Disponível em: <http://www.cs.toronto.edu/~sacook/homepage/1971.pdf.gz>.

D'AGOSTINO, M. Investigation into the complexity of some propositional calculi. D. Phil Dissertation, Programming Research Group, Oxford University. 1990.

DAVIS, G. L. M.; LOVELAND, D. A machine program for theorem proving. *Communications of the ACM*, v. 5, n. 7, p. 394–397, 1962. Disponível em: <http://-/portal.acm.org/citation.cfm?doid=368273.368557>.

DAVIS, M.; PUTNAM, H. A computer procedure for quantification theory. *In Journal of the ACM*, v. 7, p. 201–215, 1960.

E´EN, N.; RENSSON, N. S. Minisat a sat solver with conflict-clause minimization. *The International Conference on Theory and Applications of Satisfiability Testing*, 2005.

EMERSON, E. A.; CLARKE, E. M. Characterizing correctness properties of parallel programs using fixpoints. *Automata, Languages and Programming*, 1980.

FILHO, E. A. *Iniciação à Lógica matemática*. [S.l.]: Nobel, 1995.

GENTZEN, G. Untersuchungen über das logische schliessen. *Mathematische Zeitschrift*, v. 39, p. 176–210, 1969.

GOLDBERG, E.; NOVIKOV, Y. Berkmin: A fast and robust sat-solver. *Design, Automation, and Test in Europe (DATE '02)*, p. 142–149, 2002. Disponível em: <http://citeseer.ist.psu.edu/goldberg02berkmin.html>.

GROOTE, J. F.; VLIJMEN, S. van; KOORN, J. The safety guaranteeing system at station hoorn-kersenboogerd. In: *Utrecht University*. [S.l.]: IEEE, 1995. p. 57–68.

GROOTE, J. F.; WARNERS, J. P. The propositional formula checker heerhugo. *J. Autom. Reason.*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 24, n. 1-2, p. 101–125, 2000. ISSN 0168-7433.

HARRISON, J. Stalmarck's algorithm as a hol derived rule. Springer-Verlag, p. 221–234, 1996.

HINTIKKA, J. K. J. Form and content in quantification theory. *Acta Philosophica Fenicca*, VII, 1955.

IJCAR. *The International Joint Conference on Automated Reasoning*. 2010. Último Acesso: 15/04/2010. Disponível em: <http://www.ijcar.org/>.

KANGER, S. Provability in logic. *Stockholm Studies in Philosophy*, v. 1, 1957.

KLEENE, S. C. Mathematical logic. *John Wiley and Sons Inc.*, 1967.

LARRABEE, T. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, v. 1, n. 11, p. 6–2, 1992.

LEWIS M. D. T., S. T.; BECKER, B. W. Early conflict detection based bcp for sat solving. *The International Conference on Theory and Applications of Satisfiability Testing*, 2004.

MACMILLAN, K. L. Applying sat methods in unbounded symbolic model checking. *In Proceedings of 14th Conference on Computer-Aided Verification (CAV 2002)*, 2002.

MARQUES-SILVA, J. P.; GLASS, T. Combinational equivalence checking using satisfiability and recursive learning. *In Proceedings of the IEEE/ACM Design, Automation and Test in Europe*, 1999.

MARQUES-SILVA, J. P.; SAKALLAH, K. A. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, v. 48, 1999. Disponível em: <http://embedded.eecs.berkeley.edu/Alumni/wjiang/ee219b/grasp.pdf>.

MARQUES-SILVA, J. P.; SKALLAH, K. A. Boolean satisfiability in electronic design automation. *Proceedings of the 37th Annual Design Automation Conference*, p. 675–680, 2000.

MATES, B. *Lógica elementar*. [S.l.]: Editora da USP, 1968.

MONDADORI, M. An improvement if jeffrey's deductive trees. *Annali dell'Universita di Ferrara*, v. 7, 1989.

MOSKEWICZ C. F. MADIGAN, Y. Z. L. Z. M. W.; MALIK, S. Chaff: Engineering an efficient sat solver. *Proceedings of the 38th Design Automation Conference(DAC'01)*, p. 530–535, 2001. Disponível em: <http://research.microsoft.com/users/lintaoz/papers/>.

NORDSTRöM, J. Stalmarck's method versus resolution: A comparative theoretical study. Master's thesis, Stockholm University. 2001.

P=NP. *P vs NP Problem*. 2009. Último Acesso: 23/11/2009. Disponível em: <http://www.claymath.org/millennium/>.

PROF, A. et al. A theorem prover for the model evolution calculus. In: *in: IJCAR Workshop on Empirically Successful First Order Reasoning*. [S.l.]: Elsevier, 2004.

QUEILLE, J. P.; SIFAKIS, J. Specification and verification of concurrent systems in cesar. *International Symposium on Programming*, 1982.

RICHARDS, M. A tautology checker loosely related to stalmarck's algorithm. *A Seminar given at Cambridge*, 1998. Disponível em: <http://www.cl.cam.ac.uk/users/mr/chkslds-.pdf>.

RSAT-HOMEPAGE. *RSat-Homepage*. 2010. Último Acesso: 16/04/2010. Disponível em: <http://reasoning.cs.ucla.edu/rsat/>.

RYVCHIN, V.; STRICHMAN, O. Local restarts. *In proceedings of SAT*, p. 271–276, 2008.

SHEERAN, M.; BORäLV, A. Verification of arithmetic circuits using a functional language and stalmarck's method. *Technical report, Chalmers University of Technology and Prover Technology*, 1997.

SHEERAN, M.; STALMARCK, G. A tutorial on stalmarck's proof procedure. In: *In Proc. FMCAD*. [S.l.: s.n.], 1998.

SHüTTE, K. Proof theory. *Springer-Verlag*, 1977.

SMULLYAN, R. M. First order logic. *Springer-Verlag*, 1969.

STALMARCK, G. A note on the computational complexity of the pure classical implication calculus. *Information Processing Letters*, v. 31, n. 6, p. 277–278, 1989.

STALMARCK, G. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula. *Swedish Patent No. 467 076 (approved 1992), U. S. Patent No. 5 276 897 (1994), European Patent No. 0403 454 (1995)*, 1989.

STHEPAN, R. B. P.; SANGIOVANNI-VINCENTELLI, A. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design*, v. 9, n. 15, p. 1167–1176, 1996.

SYRF, P. *Use of the Stalmarck Method 1*. 1999. Último Acesso: 27/11/2009. Disponível em: <http://www-verimag.imag.fr/SYNCHRONE/SYRF% -/syrf.html>.

VELEV, M. N.; BRYANT, R. E. Effective use of boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors. *In Proceeedings of the Design Automation Conference (DAC)*, 2001.

ZCHAFF-HOMEPAGE. *zChaff-Homepage*. 2010. Último Acesso: 16/04/2010. Disponível em: <http://www.princeton.edu/~chaff/zchaff% -.html>.

ZHANG, H. Sato: An efficient propositional prover. *Proceedings of International Conference on Automated Deduction (CADE-97)*, v. 1249, p. 272–275, 1997. Disponível em: <http://citeseer.ist.psu.edu/zhang97sato.html>.

6 APÊNDICE - ÁLGEBRA DAS PROPOSIÇÕES

A seguir serão expostos alguns conceitos, que forneceram o embasamento teórico para a realização deste projeto. Tais conceitos podem ser encontrados em livros de lógica proposicional como (MATES, 1968), (ABE, 2002) e (FILHO, 1995).

6.1 Tautologia

Define-se como tautologia toda expressão proposicional, cujo resultado de sua tabela verdade encerra somente a verdade. Em outras palavras, tautologia é toda expressão proposicional cujo valor lógico é sempre a verdade, quaisquer que sejam os valores das variáveis que a compõem.

6.2 Contradição

Em contrapartida define-se como contradição toda expressão proposicional cujo resultado de sua tabela verdade encerra somente a falsidade. Em outras palavras, contradição é toda expressão proposicional cujo valor lógico é sempre falso, quaisquer que sejam os valores das variáveis que a compõem.

6.3 Contingência

Define-se como contingência toda expressão proposicional cujo resultado de sua tabela verdade possui a verdade e a falsidade, pelo menos uma vez cada. Em outras palavras, contingência é toda expressão proposicional que não é tautologia nem contradição.

6.4 Equivalência Lógica

Diz-se que uma expressão proposicional A é logicamente equivalente a outra expressão B, se os resultados das tabelas verdade destas duas proposições são idênticos. Indica-se que uma expressão A equivale a outra expressão B, pela notação: $A \equiv B$.

Em particular se as expressões A e B são ambas tautologias ou ambas contradições, então são equivalentes.

6.5 Propriedades da Conjunção

Sejam p, q e r variáveis proposicionais quaisquer. Considere V = verdadeiro e F = Falso. Prova-se pela elaboração das tabelas verdade que a conjunção, representada pelo símbolo $\wedge$, goza das seguintes propriedades:

1) Indempotente: $p \wedge p \equiv p$

2) Comutativa: $p \wedge q \equiv q \wedge p$

3) Associativa: $(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$

4) Identidade: $p \wedge V \equiv p$ e $p \wedge F \equiv F$

Tais propriedades são extensíveis a expressões compostas P, Q e R quaisquer. Sendo uma expressão composta aquela que possui mais de uma variável proposicional separadas por conectivos e/ou parênteses.

6.6 Propriedades da Disjunção

Sejam p, q e r variáveis proposicionais quaisquer. Prova-se pela elaboração das tabelas verdade que a disjunção, representada pelo símbolo $\vee$, goza das seguintes propriedades:

1) Indempotente: $p \vee p \equiv p$

2) Comutativa: $p \vee q \equiv q \vee p$

3) Associativa: $(p \vee q) \vee r \equiv p \vee (q \vee r)$

4) Identidade: $p \vee V \equiv V$ e $p \vee F \equiv p$

Tais propriedades são extensíveis a expressões compostas P, Q e R quaisquer.

6.7 Propriedades da Conjunção e da Disjunção

Baseado nas propriedades da conjunção e da disjunção tem-se como válidas as seguintes propriedades:

1) Distributiva:

$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

2) Absorção:

$p \wedge (p \vee q) \equiv p$

$p \vee (p \wedge q) \equiv p$

3) Regras DE MORGAN (1806-1871):

$\neg (p \wedge q) \equiv \neg p \vee \neg q$

$\neg (p \vee q) \equiv \neg p \wedge \neg q$

6.8 Redução do número de conectivos

Sendo ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$) os conectivos fundamentais da álgebra proposicional, pode-se demonstrar que três deles podem ser expressos em termos de apenas dois dos seguintes pares:

1) $\neg$ e $\vee$

2) $\neg$ e $\wedge$

3) $\neg$ e $\rightarrow$

Para efeito das demonstrações a seguir utilizar-se-á as equivalências:

$p \rightarrow q \equiv \neg p \vee q$

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv (\neg p \vee q) \wedge (\neg q \vee p)$

Demonstração 1

$\wedge$, $\rightarrow$, e $\leftrightarrow$ em função de $\neg$ e $\vee$:

$p \wedge q \equiv \neg\neg p \wedge \neg\neg q \equiv \neg (\neg p \vee \neg q)$

$p \rightarrow q \equiv \neg p \vee q$

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv \neg(\neg(\neg p \vee q) \vee \neg(\neg q \vee p))$

Demonstração 2

$\vee$, $\rightarrow$, e $\leftrightarrow$ em função de $\neg$ e $\wedge$:

$p \vee q \equiv \neg\neg p \vee \neg\neg q \equiv \neg\ (\neg p \wedge \neg q)$

$p \rightarrow q \equiv \neg p \vee q \equiv \neg(p \wedge \neg q)$

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv \neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q)$

Demonstração 3

$\wedge$, $\vee$, e $\leftrightarrow$ em função de $\neg$ e $\rightarrow$:

$p \wedge q \equiv \neg(\neg p \vee \neg q) \equiv \neg\ (p \rightarrow \neg q)$

$p \vee q \equiv \neg\neg p \vee q \equiv \neg p \rightarrow q$

$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p) \equiv \neg(\ (p \rightarrow q) \rightarrow \neg(q \rightarrow p))$

Note que os conectivos $\wedge$, $\vee$ e $\rightarrow$ não se exprimem em termos de $\neg$ e $\leftrightarrow$.

Outra importante dedução verificável, a partir da construção das tabelas verdade, é a "Regra da Dupla Negação": $\neg\neg p \equiv p$.

6.9 Forma Normal das Proposições

Diz-se que uma expressão proposicional encontra-se na Forma Normal (FN), se e somente se, quando muito, contém os conectivos $\neg$, $\vee$ e $\wedge$. Tome como exemplo as seguintes expressões:

- $\neg p \vee q$
- $\neg(p \wedge \neg q)$
- $(p \vee q) \wedge (\neg q \vee r)$

Toda proposição pode ser levada para uma forma normal equivalente, mediante a eliminação dos conectivos $\rightarrow$ e $\leftrightarrow$, re-escritos em função de $\neg$, $\vee$ ou $\wedge$, conforme o demonstrado anteriormente.

6.9.1 Forma Normal Conjuntiva (FNC)

Diz-se que uma expressão proposicional está na FNC, se e somente se as seguintes condições são verificadas:

- Contém, quando muito, os conectivos $\neg$, $\wedge$, $\vee$;
- O operador $\neg$ não aparece repetido (como $\neg\neg$) e não tem alcance sobre $\wedge$ e $\vee$ (isto é, só incide sobre variáveis proposicionais)
- O operador $\vee$ não tem alcance sobre $\wedge$ (isto é, não há componentes do tipo $p \vee (q \wedge r)$)

Tome como exemplo as seguintes expressões:

- $(\neg p) \vee (\neg q)$
- $(\neg p \vee q) \wedge (\neg q \vee \neg r)$

Para qualquer expressão proposicional pode-se determinar a FNC equivalente, através das seguintes transformações:

- Eliminando os conectivos $\rightarrow$ e $\leftrightarrow$ mediante a substituição de $p \rightarrow q$ por $\neg p \vee q$ e de $p \leftrightarrow q$ por $(\neg p \vee q) \wedge (p \vee \neg q)$.
- Eliminando negações repetidas e parêntesis precedidos de $\neg$ pelas regras de "Dupla Negação"e de "De Morgam".
- Substituindo $p \vee (q \wedge r)$ e $(p \wedge q) \vee r$ pelas respectivas expressões: $(p \vee q) \wedge (p \vee r)$ e $(p \vee r) \wedge (q \vee r)$. Aplicação da propriedade distributiva.

6.9.2 Forma Normal Disjuntiva (FND)

Diz-se que uma expressão proposicional está na FND, se e somente se as seguintes condições são verificadas:

- Contém, quando muito, os conectivos $\neg$, $\wedge$, $\vee$;
- O operador $\neg$ não aparece repetido (como $\neg\neg$) e não tem alcance sobre $\wedge$ e $\vee$ (isto é, só incide sobre variáveis proposicionais)

- O operador $\wedge$ não tem alcance sobre $\vee$ (isto é, não há componentes do tipo $p \wedge (q \vee r)$)

Tome como exemplo as seguintes expressões:

- $(\neg p) \wedge (\neg q)$
- $(\neg p \wedge q) \vee (\neg q \wedge \neg r \wedge p)$

Para qualquer expressão proposicional pode-se determinar a FND equivalente, através das seguintes transformações:

- Eliminando os conectivos $\rightarrow$ e $\leftrightarrow$ mediante a substituição de $p \rightarrow q$ por $\neg p \vee q$ e de $p \leftrightarrow q$ por $(\neg p \vee q) \wedge (p \vee \neg q)$.
- Eliminando negações repetidas e parêntesis precedidos de $\neg$ pelas regras de "Dupla Negação" e de "De Morgam".
- Substituindo $p \wedge (q \vee r)$ e $(p \vee q) \wedge r$ pelas respectivas expressões: $(p \wedge q) \vee (p \wedge r)$ e $(p \wedge r) \vee (q \wedge r)$. Aplicação da propriedade distributiva.

6.10 Instâncias SAT

O formato utilizado pelos solucionadores SAT é basicamente uma expressão na FNC, como a apresentada a seguir:

$f = (a) \wedge (\neg a) \wedge (\neg a \vee c \vee \neg c) \wedge (b \vee \neg c \vee \neg a) \wedge (b \vee c \vee \neg a)$

Note que a partir desta expressão, tem-se:

- Literais são definidos como uma variáveis proposicionais "x" ou sua negação "$\neg$x".
- Cláusulas são definidas pela disjunção de literais, o exemplo apresenta 5 cláusulas. Ressalta-se que o tamanho da cláusula é dado pelo número de literais presente na mesma, por exemplo as duas primeiras cláusulas do exemplo possuem tamanho 1 e as três últimas possuem tamanho 3. Cláusulas de tamanho 1 são denominadas cláusulas unitárias.
- Uma cláusula é considerada uma tautologia se tanto um literal quanto o seu complemento aparecem na cláusula, a terceira cláusula do exemplo é um exemplo de tautologia.

- Quando todos os literais de uma cláusula A estão presentes também em uma cláusula B, dize-se que a cláusula B contém A ou que A está contida em B. Cláusulas que contém outras cláusulas são consideradas redundantes. As três últimas cláusulas do exemplo se enquadram nesta situação por conterem a segunda cláusula.

- Uma cláusula é considerada redundante se, ao ser retirada, a função proposicional representada não se altera. No exemplo as três últimas cláusulas são redundantes, enquanto as duas primeiras não.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo