PROPOSITIONAL SATISFIABILITY ALGORITHMS IN EDA APPLICATIONS

by

Mukul Ranjan Prasad

Memorandum No. UCB/ERL M01/39

1 December 2001

PROPOSITIONAL SATISFIABILITY ALGORITHMS IN EDA APPLICATIONS

by

Mukul Ranjan Prasad

Memorandum No. UCB/ERL M01/39

1 December 2001

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Propositional Satisfiability Algorithms in EDA Applications

by

Mukul Ranjan Prasad

B. Tech. (Indian Institute of Technology, New Delhi) 1995

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering-Electrical Engineering and Computer Sciences

in the

GRADUATE DIVISION
of the
UNIVERSITY of CALIFORNIA at BERKELEY

Fall 2001

The dissertation of Mukul Ranjan Prasad is approved:

Chair	Date
	Date
	Date

University of California at Berkeley

2001

Propositional Satisfiability Algorithms in EDA Applications

Abstract

Propositional Satisfiability Algorithms in EDA Applications

by

Mukul Ranjan Prasad
Doctor of Philosophy in Engineering-Electrical Engineering and Computer Sciences
University of California at Berkeley
Professor Robert K Brayton, Chair

Recent years have seen a dramatic growth in the application of SAT solvers to problems in electronic design automation. This trend is due in part to recent developments in SAT algorithms which have revolutionized the field of satisfiability testing. SAT has grown from a problem of academic interest to a core computational resource of immense value.

However, despite the significant progress in this domain there is considerable room for improvement in several areas. The goal of this dissertation is to advance the theory, practice and core technology of SAT algorithms in the context of EDA applications. The success of a SAT algorithm in a given EDA application may be ensured by a realistic quantitative assessment of the projected performance of the overall algorithm in a practical setting, by carefully orchestrating the use of SAT in the application and by improving the SAT algorithm per se. This dissertation addresses these three issues.

The first part of the dissertation presents a framework for analyzing the complexity of a SAT based formulation of the combinational ATPG problem, in a practical setting. We introduce the concept of cut-width of a circuit and characterize the complexity of ATPG in terms of this property. We present theoretical results and empirical evidence to argue that a large class of practical circuits can be expected to have cut-width characteristics conducive to an efficient solution of ATPG on them. These results also help to reconcile the intractability of ATPG, as predicted by traditional worst case analysis results, with the relative ease of solving practical instances of the problem.

The second part of the dissertation focuses on the optimum orchestration of SAT methods for a given EDA application. We revisit the application of SAT algorithms to the combinational equivalence checking (CEC) problem. We argue that SAT is a more robust and flexible engine of Boolean reasoning for the CEC application than binary decision diagrams (BDDs), which have traditionally been the method of choice. Preliminary results on a simple framework for SAT-based CEC show a speedup of up to two orders of magnitude over previous methods for SAT-based CEC. Further, the prototype implementation is only moderately slower and sometimes faster than a state-of-the-art BDD-based mixed-engine commercial CEC tool.

The third and final part of the dissertation is aimed at enhancing the core techniques used in current SAT solvers. We introduce the notion of *problem symmetry* in search based SAT algorithms. We develop a *theory of essential points* to formally characterize the potential search space pruning that can be realized by exploiting problem symmetry. We unify several powerful search pruning techniques used in modern SAT solvers under a single framework, by showing them to be special cases of the theory of essential points. We also propose a new pruning rule exploiting problem symmetry. Preliminary experimental results validate the efficacy of this rule in providing additional search space pruning over the pruning realized by techniques implemented in leading-edge SAT solvers.

Professor Robert K Brayton
Dissertation Committee Chair

To Mummy & Papa ...

Contents

List of Figures vii

List of Tables viii

List of Algorithms ix

1 Introduction 1

1.1 The Propositional Satisfiability (SAT) Problem 1
1.2 SAT in EDA Applications 2
1.3 Thesis Overview & Organization 3

2 Preliminaries 8

2.1 Propositional Variables & Literals 8
2.2 Conjunctive Normal Form (CNF) Formulas 8
2.3 The Satisfiability Problem 9
2.4 The Davis-Putnam-Logemann-Loveland (DPLL) Search Algorithm for SAT 11
2.4.1 The Unit Literal Rule 12
2.4.2 The Pure Literal Rule 14
2.5 The Davis-Putnam Resolution Algorithm for SAT 14
2.6 Advanced Pruning Techniques in Search Algorithms 16
2.6.1 Conflict Based Learning 17
2.6.2 Non-chronological Backtracking 18
2.7 Transformation of Non-Clausal SAT problems to CNF 19
2.7.1 CNF Representations of Propositional Formulas 20
2.7.2 Solving SAT problems on Logic Circuits 22

3 SAT Algorithms in EDA: Recent Developments 26

3.1 SAT Algorithms & Tools 27
3.1.1 Complete SAT Solvers 27
3.1.1.1 SAT Solvers based on Propositional Formulas 27
3.1.1.2 Circuit Based Boolean Reasoning 29
3.1.1.3 Other improvements to SAT solvers 30
3.1.2 Incomplete SAT Solvers 31
3.2 SAT Applications in EDA 32

3.2.1 SAT in Verification . . . 32
3.2.2 Other Applications . . . 34
3.3 Conclusions . . . 35

4 The Practical Complexity of SAT Based ATPG **36**
4.1 Definitions and Notation . . . 38
4.2 Applying Existing Techniques . . . 41
4.2.1 Simple SAT Classes . . . 41
4.2.2 k-bounded Circuits . . . 43
4.2.3 Average-Time Analysis . . . 43
4.3 Analysis of *ATPG-SAT* . . . 46
4.3.1 Caching-Based Backtracking for CIRCUIT-SAT . . . 48
4.3.2 Cut-width and Sub-formula Count . . . 51
4.3.3 Extension to Multi-output Circuits . . . 57
4.4 Cut-width Properties of Circuits . . . 59
4.4.1 Log-bounded-width Circuits . . . 59
4.4.2 Practical VLSI circuits . . . 61
4.4.2.1 Experimental Setup . . . 61
4.4.2.2 Study of Existing Benchmark Suites . . . 63
4.4.2.3 Study of Generated Circuits . . . 63
4.4.2.4 Experimental Results . . . 65
4.5 BDDs and CIRCUIT-SAT . . . 71
4.6 Conclusions . . . 72

5 SAT-Based Combinational Equivalence Checking **74**
5.1 The Combinational Equivalence Checking Problem . . . 76
5.2 Previous Approaches . . . 78
5.3 SAT Vs. BDDs in CEC . . . 79
5.3.1 Locality and robustness of cutpoint resolution . . . 79
5.3.2 Use of previously deduced equivalences . . . 80
5.3.3 Learning more general relationships . . . 81
5.4 Proposed Methodology . . . 82
5.4.1 Segment sweeping based deduction . . . 82
5.4.2 Global hypothesis based deduction . . . 84
5.4.3 Overall Algorithm . . . 85
5.5 Experimental Results . . . 86
5.6 Conclusions and Future Directions . . . 88

6 Using Problem Symmetry to Reduce Search **91**
6.1 Definitions & Notation . . . 93
6.2 Problem Symmetry in Search . . . 94
6.3 The Theory of Essential Points . . . 97
6.4 Popular Pruning Techniques: Special Cases of Essential Point Pruning . . . 99
6.4.1 The Pure-Literal Rule . . . 99
6.4.2 Non-Chronological Backtracking (NCB) . . . 100

6.4.3 Conflict Clause Recording 102
6.5 Symmetry Based Pruning through Supercubing 103
6.5.1 Supercubing Procedure & Pruning 103
6.5.2 Proof of Correctness 104
6.5.3 Supercubing and Other Pruning Techniques 105
6.6 Experimental Results 106
6.7 Conclusions & Future Directions 108

7 Conclusions & Future Directions **110**
7.1 Future Directions 112
7.1.1 Input-distribution based complexity analysis of EDA-SAT problems 112
7.1.2 Using SAT methods in EDA problems 113
7.1.3 Improvements to SAT algorithms and tools 113

Bibliography **115**

List of Figures

1.1 SAT in a typical IC Design & Verification flow 2
1.2 Research overview 4

2.1 An Example of Non-chronological Backtracking [MSS99] 19
2.2 CNF formulas for 2-input & 1-input simple gates 25

4.1 Results of TEGUS on ATPG-SAT instances 37
4.2 Circuit C_{ψ}^{ATPG} used for ATPG-SAT 40
4.3 A simple sub-circuit C_{sub} that produces a non-q-Horn formula 42
4.4 Average Time Analysis (from [PB87]) 44
4.5 Example Circuits 47
4.6 Caching-based backtracking for Formula 4.2 50
4.7 Example cut-widths for the circuit of Figure 4.5(a) 53
4.8 Case 1 for generating injured clauses 54
4.9 Case 2 for generating injured clauses 54
4.10 Proof for ATPG circuit width vs. original circuit width 57
4.11 Example cut-width, ATPG circuit of Figure 4.5(b) 58
4.12 Cut-width results for benchmark circuits 64
4.13 Cut-width results for generated circuits 66
4.14 Cut-width results for generated circuits 67
4.15 Normalized cut-width results 70

5.1 False Negatives can be resolved by local BCP 80
5.2 Previously deduced equivalences as shallow witnesses of false negatives . . . 81

6.1 Illustration of Symmetry in Search 92
6.2 Example of Symmetry in SAT on circuits 95
6.3 Example of Symmetry in backtrack search 96

List of Tables

2.1 The disjunction operator over $\{0, 1, X\}$ 10
2.2 The conjunction operator over $\{0, 1, X\}$ 11
2.3 CNF formulas for simple gates 24

4.1 Sum of squared errors for various functional fits on each data set (Normalized by a factor of 10^5) 68

5.1 Verifying original vs. irredundant circuits 87
5.2 Verifying original vs. optimized circuits 88

6.1 Experimental results with Supercubing 107

List of Algorithms

2.1 DPLL Algorithm for SAT 13
2.2 BCP(A_{curr}) 14
2.3 DP_Resolve_Variable(ϕ, x) 15
4.1 Satisfiability through Caching-Based Backtracking 49
5.1 Global Hypothesis Deduction 85

Acknowledgements

First and foremost I would like to thank my research advisor Prof. Robert K Brayton for his invaluable support and guidance. Bob continues to inspire me with his dedication, perseverance, integrity and professionalism. I am grateful to him for giving me the freedom to choose a research area best suited to my interests and aptitude and for moulding my research efforts into a meaningful contribution.

I am grateful to Prof. Alberto Sangiovanni-Vincentelli for his support and encouragement during my stay at Berkeley. I have had the good fortune of interacting with him in several capacities and have immensely benefited from this association. I would also like to thank Prof. Kurt Keutzer for his constructive feedback as chair of my qualifying examination committee. The research presented in this dissertation was seeded through a class project under his guidance.

I would also like to thank Prof. Ilan Adler and Prof. Shmuel Oren for their insightful comments and feedback as members of my dissertation committee and qualifying examination committee respectively.

My research efforts at Berkeley have given me the opportunity to interact and collaborate with several bright individuals. A special word of gratitude for Eugene Goldberg who has worked with me as my mentor, colleague and friend over the last two years. The research presented in Chapters 5 and 6 is a result of collaborative work with him. My close interaction with him has helped me appreciate the value of unconventional and fundamental thinking. I would like to thank Philip Chong for his contribution to the work presented in Chapter 4. I have been fortunate to collaborate with Gurmeet Singh Manku, Desmond Kirkpatrick, Sunil Khatri, Amit Mehrotra and Freddy Mang on various other research problems, although that research is not included in this dissertation. I would also like to acknowledge Prof. João Poalo Marques-Silva of the Technical University of Lisbon and Dr. Andreas Kuehlmann of Cadence Berkeley Labs. for the many interesting and fruitful discussions I have had with them regarding my research.

The CAD group at Berkeley provided a stimulating and friendly environment for me during my student life at Berkeley. Several individuals in this group have contributed to making my stay here a memorable one. I would like to thank Rajeev Murgai, Harry Hsieh, Shaz Qadeer, Sriram Rajamani, Premal Buch, Amit Narayan, Alok Agrawal, Sriram Krishnan, Adnan Aziz, Edoardo Charbon and Yuji Kukimoto for their constructive advice and guidance, my cubicle-mates Mark Spiller, Claudio Pinello and Marco Sgroi for putting up with me and Lixin Su, Wilsin Gosti and Luca Carloni for their support and company. I would also like to thank my friends and colleagues Kaustav Banerjee, Rizwan Rahmani, Kiran and Jawahar Jain for their tangible and intangible contributions to both personal and professional aspects of my life at Berkeley.

I am grateful to the administrative and support staff in the EECS department at Berkeley for promptly and efficiently attending to my needs over the years. Ruth Gjerde, Brad Krebs, Flora Oviedo and Diane Chang deserve a special word of thanks for their assistance.

The research presented in this dissertation was supported by the California MICRO program and Cadence Berkeley Laboratories. Their generous support is gratefully acknowledged.

Finally, I would like to thank my family for their love and support. My father Dr. Girish Prasad and my mother Prabha Prasad have always been my role models and sources of inspiration. It was they who encouraged me to pursue a career in engineering and to come to the US to further my aspirations. Their love, guidance, sacrifice, support and unflinching belief in my capabilities have always helped me surmount the toughest of challenges with ease. I owe all my achievements, past, present and future to them. The past year has been the most challenging and rewarding periods of my professional career, thus far. I would like to thank my wife Shuchi for her love, encouragement, sacrifice and support which helped me get through this time and come out stronger. I would also like to thank my sister and brother-in-law Meenakshi and Atul Garg, my parents-in-law Prof. R.C. Jain and Saroj Jain and the rest of my extended family in the San Francisco Bay area for their love and support over the years. Last but not least I would like to thank God almighty.

Chapter 1

Introduction

1.1 The Propositional Satisfiability (SAT) Problem

The *Propositional Satisfiability (SAT)* problem is a core problem in mathematical logic and computing theory. The SAT problem is known to be *NP-Complete* [GJ79], one of the first problems to be proven NP-Complete. Thus it is generally accepted that any algorithmic technique for solving an arbitrary instance of SAT could require, in the worst-case, computational resources exponential in the size of the problem. In other words it is highly unlikely that there exists an algorithm for SAT that can solve an arbitrary instance of the problem, *efficiently*[1].

Over the years, several interesting problems from various applications in logic and computer science have been modeled as satisfiability problems. Moreover, several instances of such problems, arising in practise, can be efficiently solved by SAT solvers. Thus, any advance in SAT solver technology may translate into significant improvements in several practical applications. This potential coupled with the simple formulation of the satisfiability problem and its inherent intractability has continued to interest theoreticians and practitioners alike, over the last four decades.

There have been significant advancements in SAT solver technology since the first complete algorithm for Boolean Satisfiability proposed by Davis and Putnam [DP60] in 1960. In current practice SAT is fundamental in solving several problems in areas as diverse as automated reasoning, computer-aided manufacturing, machine vision, databases, robotics,

[1]By efficiently we mean that the algorithm uses computational resources polynomial in the size of the problem.

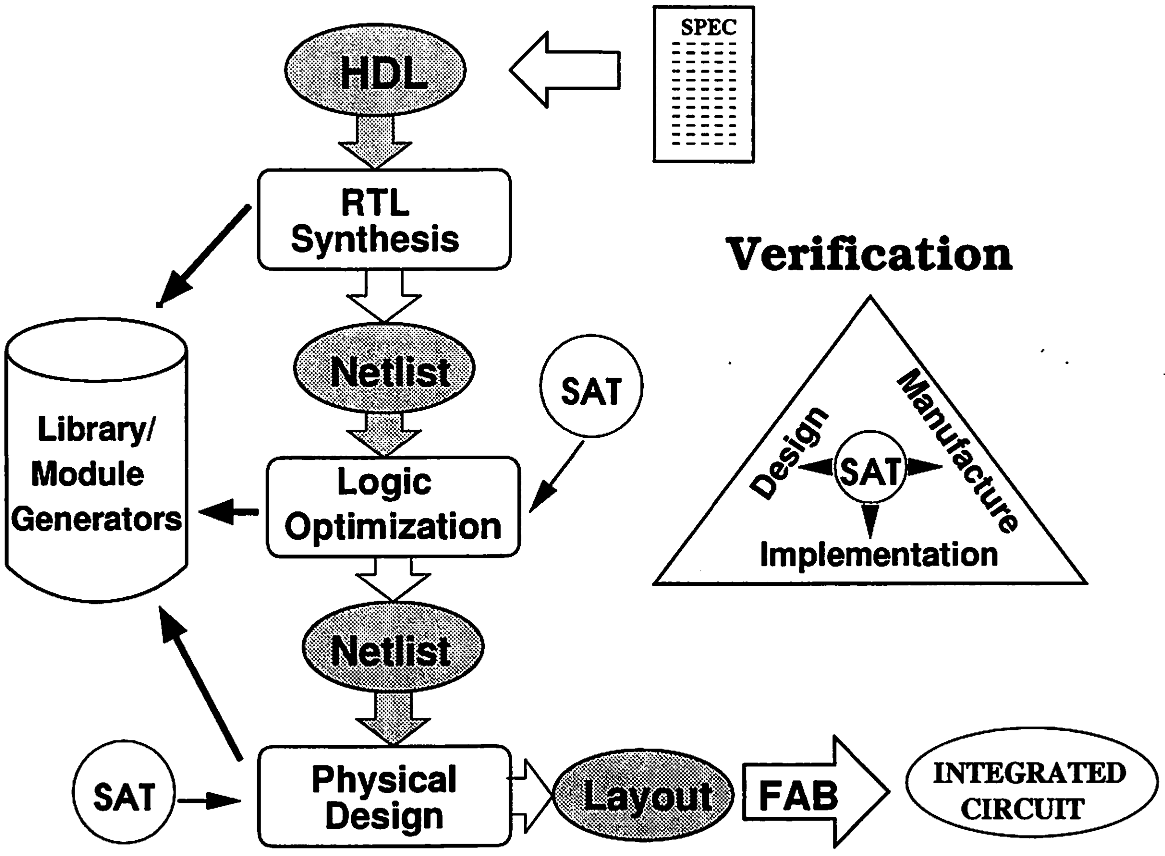


Figure 1.1: SAT in a typical IC Design & Verification flow

computer architecture and computer network design [GPFW97]. This ever-widening gamut of SAT applications continues to fuel further research in this area.

1.2 SAT in EDA Applications

The design and manufacture of integrated circuits is a complex process. Its richness and complexity translates into a number of challenging problems in electronic design automation which offers a fertile ground for application of optimization and decision procedures such as Boolean Satisfiability.

The use of SAT procedures for EDA applications is a relatively recent phenomenon. While branch and bound procedures, somewhat similar (although not nearly as powerful) to those used in SAT solvers, have been employed in EDA tools for some time now, the application of SAT solvers per se to EDA problems is only a decade old.

Recent years have seen dramatic improvements in SAT algorithms and tools [Zha97, MSS99, MMZ+01], allowing much larger problem instances to be solved. It is not uncommon for current leading-edge SAT solvers to efficiently handle problem instances with thousands of variables and tens of thousands of clauses [MMZ+01]. This has expanded the realm of applicability of SAT solvers in EDA and fueled the growth of SAT from a problem, primarily of academic interest, to an enabling technology for several EDA applications. Spurred by these recent advancements SAT algorithms have been successfully applied to solve problems from a wide variety of EDA applications such as Automatic Test Pattern Generation (ATPG), timing analysis, sequential and combinational logic optimization, crosstalk noise analysis, FPGA routing, functional vector generation, combinational equivalence checking, reachability analysis, model checking and microprocessor verification [MSS00].

Figure 1.1 shows the potential role of SAT in a typical design and verification flow for integrated circuits. It shows that SAT formulations can be employed in almost all major steps of the design process and in all aspects of verification *i.e.* design, implementation and manufacture verification. Despite this significant progress much ground remains to be covered in several areas.

The current level of understanding regarding the strengths and weaknesses of SAT in the context of various EDA problem settings is limited. A deeper theoretical as well practical understanding of these issues may lead to more effective ways of using SAT for EDA problems. This has been an impediment in making SAT the method of choice for certain applications or having it work with more established engines such as structural ATPG and BDD methods. The other aspect is of course to develop more effective core SAT methods to expand the scope of SAT to emerging EDA applications.

1.3 Thesis Overview & Organization

The research described in this dissertation is aimed at advancing the theory, practice and core technology of SAT algorithms in the context of EDA applications. Figure 1.2 shows the general setting in which this research is based. The scenario is that of a SAT solver being applied to solve or aid in the solution of EDA applications such as combinational verification, ATPG *etc.* The various components of this dissertation focus on different aspects of enhancing the efficacy of such a combination. Specifically, this work addresses

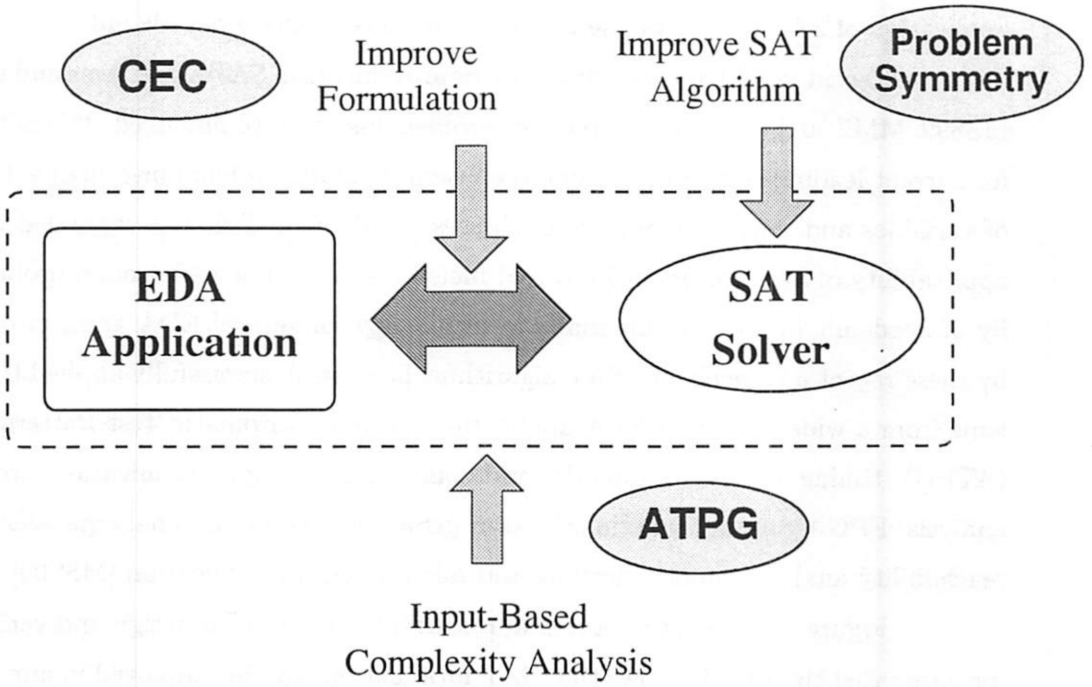


Figure 1.2: Research overview

the following three aspects:

1. **Input Distribution Dependent Complexity Analysis:** The first step in trying to solve a given application through a certain algorithm is to perform a formal quantitative analysis of the complexity of the problem and the expected performance of the algorithm on the given problem[2]. In practice, and specifically in the EDA community, the kind of performance analysis that is done is a worst case complexity analysis. By definition, it is pessimistic. This fact assumes greater relevance in our context, *i.e.* SAT used in an EDA application. SAT is known to be NP-Complete [GJ79] and probably the EDA application it is being used in is also NP-Complete or even harder[3]. Thus worst case complexity analysis might imply that there is little hope of any practical success. The reality is quite the contrary. The answer lies in the fact that SAT instances coming from EDA applications are a specialized and structured subset of the general class of SAT instances. Traditional worst-case complexity cannot make

[2]These two entities are not identical since the problem may have an inherent complexity different from the algorithm employed to solve it.

[3]Such is the case with most EDA problems.

this distinction. Therefore, what is required is an analysis that models and accounts for the characteristics of SAT instances arising from EDA applications, in a practical setting.

2. **SAT formulation:** A SAT formulation of a problem determines "how" the solver is being used to solve the application at hand. The formulation determines the number, the nature and the specific CNF representation of SAT instances generated by a single instance of the EDA application as well as the sequence in which these instances are solved by a SAT solver. For example, a single instance of combinational ATPG[4] is typically formulated as a single SAT instance [Lar92] whereas a single instance of a Bounded Model Checking application [BCCZ99] produces a set of SAT instances. In combinational ATPG, the specific CNF representation is key. It has been shown [Lar92, SBSV96] that the addition of fault propagation information to the CNF in the form of clauses known as *active clauses* can have a dramatic impact to the ease of solving the SAT instance, and hence the solvability of the original ATPG instance. The formulation also determines the configuration of the SAT solver such as the variable ordering heuristic, the information (if any) passed between the SAT solver and other components of the application tool *etc.* Usually, the formulation stems from insights about the EDA application and empirical guidelines from what works in practice.

3. **The SAT Solver:** Another obvious mode of enhancing the efficiency of an EDA application is to improve the SAT solver itself. This may be general enhancements to the SAT algorithm which may benefit any application using such a solver. For example, recent developments in SAT algorithms such as *conflict based learning* and *non-chronological backtracking* [MSS99] have been shown to be beneficial in most SAT applications. Alternatively, these enhancements could be algorithmic features that more directly benefit the specific application at hand, *e.g.* a recursive learning enhanced SAT solver aimed at learning equivalences [MSG99] is particularly useful for combinational verification but not so effective in other EDA applications.

This dissertation is organized as follows. Chapter 2 develops the notational framework used in the rest of the dissertation. It also presents some fundamental SAT algorithms

[4]A single instance refers to the task of testing a particular single stuck-at fault in the circuit

and some recently proposed pruning techniques. These form the foundation on which most of the theoretical results and algorithms developed in this research are based.

Chapter 3 surveys some more recent developments in SAT algorithms and applications, particularly the ones relevant to EDA. These developments have acted as catalysts for SAT research in EDA. Comprehensive surveys on some of the earlier developments on these subjects can be found in [GPFW97] and [MSS00].

The research contribution of this dissertation is organized in three chapters, each addressing one of the three axes of contribution described above.

The first part of the research, presented in Chapter 4 is an attempt at input distribution based complexity analysis of an EDA SAT application. The problem analyzed is a SAT based formulation of the *combinational ATPG* problem. The results shed light on the following paradox. Empirical observation shows that practically encountered instances of combinational ATPG are efficiently solvable. However, it has been known for more than two decades that ATPG is an NP-complete problem [IS75]. The presented analysis is one of the first attempts to reconcile these results. We introduce the concept of *cut-width* of a circuit and characterize the complexity of SAT based ATPG in terms of this property. We introduce the class of *log-bounded width* circuits and prove that SAT based combinational ATPG is efficiently solvable on members of this class. The class of *log-bounded width* circuits is shown to strictly subsume the class of *k-bounded circuits* introduced in [Fuj88]. We provide empirical evidence which indicates that an interestingly large class of practical circuits is expected to have *log-bounded width.* The analysis framework presented is fairly general and could be used to perform a similar analysis of other SAT-EDA problems.

Chapter 5 presents a SAT based algorithmic framework to address the *combinational equivalence checking (CEC)* problem. This addresses the second aspect described above *i.e.* developing improved methods of using SAT in a typical EDA application. The contributions of this work are twofold. First, it offers a qualitative understanding of aspects or parts of the CEC problem that are best suited to be handled by a SAT solver rather than BDDs, which currently form the computational core of most CEC tools. Second, it develops improved methods of using SAT algorithms for CEC which dramatically outperform previous approaches for SAT based CEC.

Chapter 6 addresses improvements to the core technology of SAT solvers. Specifically, we introduce the notion of *problem symmetry* in search-based SAT algorithms. A *theory of essential points* is developed to characterize the potential search-space pruning that can be realized by exploiting problem symmetry. Several search-pruning techniques used in modern SAT solvers are shown to be special cases of the general theory of essential points. We also propose a new pruning rule exploiting problem symmetry. Preliminary experimental results validate the efficacy of this rule in providing additional search-space pruning beyond the pruning realized by techniques implemented in current leading-edge SAT solvers.

Chapter 7 summarizes the main results and conclusions of this research and presents avenues for improvement and directions for future research.

Chapter 2

Preliminaries

2.1 Propositional Variables & Literals

Throughout this dissertation, *propositional variables* (interchangeably referred to as *Boolean variables* or just *variables*) will be denoted by symbols drawn from the set of symbols $[rstuwxyz]_{[0-9]^*}$. A *propositional variable*, x can assume a *logic value*, denoted $\nu(x)$, with $\nu(x) \in \{0, 1, X\}$. When $\nu(x) = 1$ (the proposition is **TRUE**) or $\nu(x) = 0$ (the proposition is **FALSE**), x is said to be *assigned* and when $\nu(x) = X$, x is said to be *unassigned*. X is also used to denote that the value of the variable is either *unknown* or *undecided*.

A *literal*, l, is an instance of a variable or its complement. *e.g.* if x is a propositional variable, $\overline{x}$ and x represent, respectively the complemented and un-complemented literals derived from it, *i.e.* the propositions $x = 0$ and $x = 1$ respectively.

The following treatment will assume a set of variables $V = \{x_1, x_2, \ldots, x_N\}$, the cardinality of which is N or simply $|V|$.

2.2 Conjunctive Normal Form (CNF) Formulas

A *conjunctive normal form (CNF)* is comprised of clauses. A *clause*, ω is defined as a disjunction of literals,

$$\omega = \sum_{i=1}^{|\omega|} l_i \tag{2.1}$$

where each l_i is a literal and the clause ω is comprised of $|\omega|$ literals. Alternatively, a clause can be represented as a set of literals, $\omega = \{l_1, l_2, \dots, l_{|\omega|}\}$.

A *CNF formula* ϕ is defined as a conjunction of clauses,

$$\phi = \prod_{i=1}^{|\phi|} \omega_i \tag{2.2}$$

where each ω_i is a clause and the CNF formula ϕ is comprised of $|\phi|$ clauses. Alternatively, a CNF can be represented as a set of clauses, $\phi = \{\omega_1, \omega_2, \dots, \omega_{|\phi|}\}$.

Example 2.1 *An example of a CNF formula is* $\phi_1 = (x_1 + x_2 + \overline{x_3}) \cdot (\overline{x_1} + x_3) \cdot (x_1 + \overline{x_2} + x_3)$*. Alternatively, in set notation* $\phi = \{\omega_1, \omega_2, \omega_3\}$*, where* $\omega_1 = \{x_1, x_2, \overline{x_3}\}$*,* $\omega_2 = \{\overline{x_1}, x_3\}$ *and* $\omega_3 = \{x_1, \overline{x_2}, x_3\}$.

The CNF of Example 2.1 is based on a set of three propositional variables, x_1, x_2, x_3 and is comprised of three clauses, ω_1, ω_2 and ω_3. Clause ω_3 is composed of three literals, $x_1, \overline{x_2}$ and x_3.

2.3 The Satisfiability Problem

Given a set of variables V, an *assignment* A is a function that maps a set $U \subseteq V$ to $\{0, 1\}$. A is interpreted as a set $A \subseteq U \times \{0, 1\}$ of variable value pairs (x, ν_x). A variable $y \notin U$, not assigned a value by A assumes a value X by default. Assignment A is said to be *complete* if every variable $x \in V$ is assigned a value by A, *i.e.* $|A| = |V|$, otherwise A is said to be *partial* ($|A| < |V|$).

Given an assignment V (partial or complete), and a literal l_y of variable y, the value of l_y under V, denoted $l_y\,|_A$ is given by

$$\begin{aligned} l_y|_A &= \nu_y && \text{if } l_y = y \text{ and } (y, \nu_y) \in A \\ &= \neg\nu_y && \text{if } l_y = \overline{y} \text{ and } (y, \nu_y) \in A \\ &= X && \text{otherwise} \end{aligned} \tag{2.3}$$

Given a clause ω the value of ω under assignment A is denoted by $\omega\,|_A$ and computed as:

$$\omega\,|_A = \sum_{i=1}^{|\omega|} l_i\,|_A$$

x	y	$x+y$
0	0	0
0	1	1
0	X	X
1	0	1
1	1	1
1	X	1
X	0	X
X	1	1
X	X	X

Table 2.1: The disjunction operator over $\{0, 1, X\}$

The definition of the disjunction operator ($+$ or $\vee$) over $\{0, 1, X\}$ is given by Table 2.1. If $\omega|_A = 1$ then ω is said to be ***satisfied***. If $\omega|_A = 0$ then ω is said to be ***unsatisfied***. Otherwise ω is said to be ***unresolved***. The unassigned literals of ω, if any, are called ***free literals***. An unresolved clause with only one free literal is called a ***unit clause***. The ***empty clause ()*** or ***contradiction*** is a clause with no literals (or all literals unsatisfied and removed) and is a shorthand representation for the unsatisfiable clause (0).

The value of a CNF formula, ϕ (defined as per Equation 2.2) is denoted by $\phi|_A$ and computed as

$$\phi|_A = \prod_{j=1}^{|\phi|} \omega_j|_A$$

The definition of the conjunction operator ($\cdot$ or $\wedge$) over $\{0, 1, X\}$ is given by Table 2.2. If $\phi|_A = 1$ then ϕ is said to be ***satisfied***. If $\phi|_A = 0$ then ϕ is said to be ***unsatisfied***. Otherwise ϕ is said to be ***unresolved***. It is easily verified that a CNF, ϕ is satisfied under assignment A if and only if at least one literal in each clause of ϕ assumes value 1 under A.

Definition 2.1 (The SAT Problem) *Given a CNF formula ϕ, the Satisfiability problem posed on ϕ, SAT(ϕ) seeks to determine if there exists an assignment A under which ϕ is satisfied. Such an assignment, if one exists, is called a* satisfying assignment *for ϕ and the formula ϕ is termed* satisfiable. *Otherwise ϕ is termed* unsatisfiable.

It can be verified that the formula ϕ_1 of Example 2.1 is satisfiable and that the assignment $A = \{(x_1, 1), (x_3, 1)\}$ is a satisfying assignment for ϕ_1.

x	y	$x \cdot y$
0	0	0
0	1	0
0	X	0
1	0	0
1	1	1
1	X	X
X	0	0
X	1	X
X	X	X

Table 2.2: The conjunction operator over $\{0, 1, X\}$

2.4 The Davis-Putnam-Logemann-Loveland (DPLL) Search Algorithm for SAT

The *Davis-Putnam-Logemann-Loveland* or *DPLL* algorithm [DLL62][1] is a search based algorithm for SAT. It is one of the earliest complete algorithms for the Satisfiability problem and also the backbone of almost all successful SAT solvers. A brief description of the algorithm is presented in the following. It is, in essence, very similar to the original version that appeared in [DLL62] but specifically based on current implementations of the DPLL algorithm.

The DPLL algorithm for SAT is a branch and bound search algorithm. Given a CNF ϕ based on the variable set V the algorithm searches all possible partial assignments to V for a satisfying assignment. The search terminates either on finding a satisfying assignment or after all partial assignments have been exhausted.

The search tree is organized by exploring possible extensions of the current partial assignment, A_{curr}. An unassigned variable, y is chosen and A_{curr} is extended by branching on the two disjoint possibilities $A_{curr} \cup (y, 0)$ and $A_{curr} \cup (y, 1)$. The variable y is called a *decision variable* and an assignment made to it, a *decision assignment.* The search is pruned whenever the current partial assignment results in a clause of ϕ being unsatisfied (*i.e.* $\phi|_{A_{curr}} = 0$). Such a situation is termed a *conflict condition* or simply a *conflict* and denoted by $\mathcal{X}$. The particular clause unsatisfied under $\mathcal{X}$, through which the conflict was detected,

[1]H. Putnam is not one of the authors of the article proposing the algorithm but is still credited with this algorithm since the procedure draws heavily from the Davis-Putnam algorithm [DP60] which was a predecessor to this work.

is called the *conflict clause* and denoted by $\omega(\mathcal{X})$. A particular conflict $\mathcal{X}$ is identified by its conflict clause $\omega(\mathcal{X})$. A *decision level* $\lambda(x)$ is associated with each instance of a decision variable x. This is the level of the decision node in the search tree explored by the algorithm. The root of this tree has decision level 1. The search tree explored by the algorithm can be graphically represented as a tree graph, where the decision variables are the nodes and edges from these nodes, labeled with Boolean values, denote decision assignments to the respective variables. In keeping with this graphical representation, the two assignments to a decision variable x are called the *left branch* and *right branch* assignments of x, where the left branch conventionally denotes the assignment explored first by the algorithm.

The above simple branch and bound scheme is augmented with two other pruning rules, called the *unit literal rule* and the *pure literal rule*. The pseudo code for the DPLL algorithm is shown in Algorithm 2.1.

2.4.1 The Unit Literal Rule

According to this rule, if at any point in the search a unit literal clause, $\{l_x\}$ is derived then the assignment $(l_x, 1)$ can be immediately made, without having to branch on the variable x. This rule follows from the fact that the opposite assignment $(l_x, 0)$ will render the above clause, and hence ϕ, unsatisfied. Obviously there is no solution for **SAT**(ϕ) in that subspace. Hence that subspace can be pruned away. Iterated application of the Unit Literal rule is called *Boolean Constraint Propagation (BCP)* and is a powerful component of all DPLL based SAT algorithms. The pseudo code for BCP appears in Algorithm 2.2.

Variable assignments derived through BCP are referred to as *deduced assignments* or *implied assignments* (also *implications* or *deductions* in short). With each deduced variable y (and the corresponding deduced assignment l) it is convenient to associate a clause, $\omega_{\mathcal{I}}(y)$ from which the implication was derived. As with decision variables, deduced variables are also assigned a decision level $\lambda(y)$ which is recursively defined as:

$$\lambda(y) = \max_{x \in \omega_{\mathcal{I}}(y), x \neq y} \lambda(x)$$

It is easy to show that under a given partial assignment, $\lambda(x)$ is unambiguously defined for each assigned variable x by the above definitions. Henceforth, the notation $x = \nu @ \lambda$ will be used to denote that variable x is assigned a value ν at decision level λ. Occasionally, it is useful to annotate a set of assignments with decision levels for each assignment, using the above notation.

Algorithm 2.1 DPLL Algorithm for SAT

```
procedure DPLL_main($\phi$)
$A_{curr} \leftarrow \emptyset$
$x \leftarrow \emptyset$
$\nu_x \leftarrow$ "NULL"
return DPLL_recursive($\phi, A_{curr}, x, \nu_x$)

procedure DPLL_recursive($\phi, A_{curr}, x, \nu_x$)
$A_{curr} \leftarrow A_{curr} \cup (x, \nu_x)$                // Extend $A_{curr}$ by setting $x \leftarrow \nu_x$
if All clauses satisfied then
  return "SAT"
else if Exists an unsatisfied clause $\omega_{unsat}$ then
  return "UNSAT"
else
  status $\leftarrow$ "OPEN"
end if
bcp_status $\leftarrow$ BCP($A_{curr}$)
if bcp_status $\neq$ "OPEN" then
  return bcp_status
end if
Apply_PureLiteral_Rule( $A_{curr}$)
if All clauses satisfied then
  return "SAT"
end if
$x_{next} \leftarrow$ Choose_Next_Variable($\phi, A_{curr}$)
left_branch_status $\leftarrow$ DPLL_recursive( $\phi, A_{curr}, x_{next}, 0$)
if left_branch_status = "SAT" then
  return "SAT"
else
  right_branch_status $\leftarrow$ DPLL_recursive($\phi, A_{curr}, x_{next}, 1$)
  if right_branch_status = "SAT" then
    return "SAT"
  else
    return "UNSAT"                  // Both branches of $x_{next}$ unsatisfiable
  end if
end if
```

Algorithm 2.2 BCP(A_{curr})

```
status ← "OPEN"
while unit clauses in $\phi$ AND status == OPEN do
  if exists unit clause $\omega_{unit} = l_x$ then
    $A_{curr} \leftarrow A_{curr} \cup (l_x, 1)$          // Make unit literal assignment
  end if
  if exists an unsatisfied clause $\omega_{unsat}$ then
    status ← "UNSAT"
  else if all clauses satisfied by $A_{curr}$ then
    status ← "SAT"
  end if
end while
return status
```

2.4.2 The Pure Literal Rule

The pure literal rule states that if a variable x appears only as literals of one polarity, say l_x in the currently unresolved clauses of the CNF ϕ then the assignment $(l_x, 1)$ can be immediately effected, without needing to branch on the variable x.

This rule is based on the result that if a solution for **SAT**(ϕ) lies in the subspace of the current partial assignment, A_{curr} extended with the assignment $(l_x, 0)$ then there must be a solution for **SAT**(ϕ) in the subspace $A_{curr} \cup (l_x, 1)$. Thus from the point of view of testing satisfiability of ϕ it is sufficient to explore just the latter subspace.

2.5 The Davis-Putnam Resolution Algorithm for SAT

The *Davis-Putnam algorithm* [DP60] was one of the first complete algorithms for CNF satisfiability. The basic operation used in this algorithm is the *consensus* [Qui55] operation over clauses. Consider clauses ω_1 and ω_2 which contain a literal of variable x, instantiated in opposite polarities in ω_1 and ω_2. Say, $\omega_1 = (\omega_1' + x)$ and $\omega_2 = (\omega_2' + \overline{x})$. Then the consensus of ω_1 and ω_2 with respect to variable x, denoted $c(\omega_1, \omega_2, x)$, is defined as

$$c(\omega_1, \omega_2, x) = \omega_1' + \omega_2' \tag{2.4}$$

In theorem proving terminology [Rob65, Lov78] consensus over clauses is commonly referred to as *ground resolution* or simply *resolution*. Resolution and its derivatives form an integral component of theorem proving algorithms.

Algorithm 2.3 DP_Resolve_Variable(ϕ, x)

```
$\phi_x \leftarrow \{\omega | x \in \omega \vee \overline{x} \in \omega\}$                // clauses with literal $x$ or $\overline{x}$
$\varphi_1 \leftarrow \{\omega - \{x\} | \omega \in \phi \wedge \{x\} \in \omega\}$          // $x$-literal clauses with $x$ dropped
$\varphi_0 \leftarrow \{\omega - \{\overline{x}\} | \omega \in \phi \wedge \{\overline{x}\} \in \omega\}$          // $\overline{x}$-literal clauses with $\overline{x}$ dropped
$\phi_x^r \leftarrow \{\omega_0 \cup \omega_1 | \omega_0 \in \varphi_0, \omega_1 \in \varphi_1\}$          // resolve out variable $x$
$\phi \leftarrow (\phi - \phi_x) \cup \phi_x^r$
return $\phi$
```

The Davis-Putnam procedure is an iterative algorithm. Variables of the CNF ϕ are resolved out one at a time, as per Algorithm 2.3, until the empty clause is derived or the formula becomes a *tautology*. The former condition happens when two unit-clauses of opposite polarity (*e.g.* $\{x\}$ and $\{\overline{x}\}$) are resolved and indicates that the original CNF ϕ is unsatisfiable. The latter condition occurs when all clauses are resolved out without encountering an empty clause and indicates that ϕ is satisfiable.

This basic procedure is augmented with a few other rules to simplify the CNF, particularly the *pure literal rule* and the *unit clause rule*. These rules, discussed earlier (Sections 2.4.2 and 2.4.1 respectively) were first proposed in the Davis-Putnam algorithm [DP60] and later adapted for use in the DPLL procedure [DLL62]. Simply put, using the *pure literal rule* and the *unit clause rule*[2] in the DP procedure amounts to applying the variable resolution procedure (Algorithm 2.3) on pure-literal variables and unit-literal variables before other variables.

The biggest and most obvious drawback of the Davis-Putnam algorithm is that it is fairly space intensive. The DPLL backtracking algorithm, whose space complexity is linear in the size of the SAT instance, is the preferred alternative for most practical SAT solvers. Another minor drawback of the Davis-Putnam algorithm is that when it returns with the answer that the CNF is satisfiable, a witness satisfying assignment is not provided[3]. From a practical point of view such an assignment could be important in the target application

[2]To be precise, the unit literal rule is a two step procedure of first resolving out the variable appearing in the unit literal and then simplifying the new clauses generated by resolution through *clause subsumption*.

[3]The procedure can be augmented to derive such an assignment albeit at the cost of an additional computational overhead.

from which the SAT instance was derived. *e.g.* a test vector in the case of SAT-based ATPG. In DPLL based SAT, satisfying assignments are free by-products of the algorithm itself.

2.6 Advanced Pruning Techniques in Search Algorithms

Much of the success of SAT solvers in EDA applications can be attributed to recent advancements in search pruning techniques. Two prominent contributions in this area are *conflict based learning* and *non-chronological backtracking*. These concepts had been used in constraint satisfaction problem (CSP) solvers [Dec90, SV93] but were adapted to the propositional satisfiability problem and popularized by the **RELSAT** [BS97] and **GRASP** [MSS99] SAT solvers.

Central to both these techniques is the notion of *conflict analysis*. Given a conflict condition $\mathcal{X}$ encountered by the SAT algorithm during the search and the current set of assignments A_{curr}, conflict analysis seeks to determine a subset $A_{\mathcal{R}}(\mathcal{X}) \subseteq A_{curr}$ which can be held responsible for the conflict $\mathcal{X}$. Specifically, the assignments $A_{\mathcal{R}}(\mathcal{X})$ should be such that if just these assignments are made on the CNF ϕ (accompanied by the ensuing BCP) then the original conflict clause $\omega(\mathcal{X})$ of conflict $\mathcal{X}$ would still be unsatisfied without having made the remaining assignments of A_{curr}. What makes this analysis important is the observation that in practice only a small fraction of the assignments leading up to a conflict actually influence or cause the conflict. The following example taken from [MSS99] illustrates the notion of conflict analysis.

Example 2.2 *[MSS99] Consider the following set of clauses that form part of a CNF ϕ,*

$$\begin{array}{rclrcl}
\omega_1 & = & (\overline{x_1} + x_2) & \omega_6 & = & (\overline{x_5} + \overline{x_6}) \\
\omega_2 & = & (\overline{x_1} + x_3 + x_9) & \omega_7 & = & (x_1 + x_7 + \overline{x_{12}}) \\
\omega_3 & = & (\overline{x_2} + \overline{x_3} + x_4) & \omega_8 & = & (x_1 + x_8) \\
\omega_4 & = & (\overline{x_4} + x_5 + x_{10}) & \omega_9 & = & (\overline{x_7} + \overline{x_8} + \overline{x_{13}}) \\
\omega_5 & = & (\overline{x_4} + x_6 + x_{11}) & & &
\end{array}$$

Suppose the current assignment is given by

$$A_{curr} = \{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2, \dots\}$$

Now make the decision assignment $x_1 = 1@6$. *This decision assignment and ensuing BCP implications produce a conflict* $\mathcal{X}_1$ *on the given clauses. The specific conflict analysis algorithm presented in [MSS99] when applied to this example produces*

$$A_{\mathcal{R}}(\mathcal{X}_1) = \{x_1 = 1@6, x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3\}$$

Thus this conflict does not depend on any decisions or implications made at decision levels $2, 4$ *or* 5.

Conflict analysis is performed by analyzing the chronology of decision and deduced assignments leading up to the conflict, to determine an appropriate $A_{\mathcal{R}}(\mathcal{X})$. A given conflict $\mathcal{X}$ can have several possible $A_{\mathcal{R}}(\mathcal{X})$ which can act as sufficient reasons for that conflict [MSS99]. Depending on the specific nature of the conflict analysis procedure, one or more specific $A_{\mathcal{R}}(\mathcal{X})$ sets out of the many possibilities may be examined on a given conflict. While it is generally accepted that the particular choice (or choices) of $A_{\mathcal{R}}(\mathcal{X})$ can have a significant impact on the performance of the SAT algorithm the problem of determining the most suitable $A_{\mathcal{R}}(\mathcal{X})$ for a given $\mathcal{X}$ (both in terms of the efficiency of computing $A_{\mathcal{R}}(\mathcal{X})$ and the potential search space pruning it can eventually provide) remains an open question. Recent work by Zhang *et al.* [ZMMM01] addresses this problem and provides some empirical guidelines. However, much work remains to be done on both the theoretical and practical aspects of this problem.

2.6.1 Conflict Based Learning

Conflict based learning is the notion of recording information, on encountering a conflict condition, with the objective of using the recorded information to avoid future occurrences of the same or related conflicts. The recorded information is an implicate of the CNF ϕ which is unsatisfied under the given conflict condition. A trivial candidate for this purpose would be the conflict clause $\omega(\mathcal{X})$. However, this clause does not add any additional value to the CNF since it is already a part of ϕ. Therefore, the added clause, $\omega_{\mathcal{R}}(\mathcal{X})$ is a clause derived from the set $A_{\mathcal{R}}(\mathcal{X})$ obtained through conflict analysis. Specifically,

$$\omega_{\mathcal{R}}(\mathcal{X}) = \bigvee_{l \in A_{\mathcal{R}}(\mathcal{X})} \bar{l} \tag{2.5}$$

Hence conflict based learning is also known as *conflict clause recording* or simply *clause recording*. For the conflict $\mathcal{X}_1$ of Example 2.2 the recorded clause would be $\omega_{\mathcal{R}}(\mathcal{X}_1) =$

$(\overline{x_1} + x_9 + x_{10} + x_{11})$. From the definition of conflict analysis, it follows that adding $\omega_{\mathcal{R}}(\mathcal{X})$ to ϕ ensures that the search will not regenerate the conflicting assignment that led to $\mathcal{X}$. Further, adding $\omega_{\mathcal{R}}(\mathcal{X})$ has the potential of identifying future implications that are not derivable otherwise.

2.6.2 Non-chronological Backtracking

Backtracking, in the context of a search based SAT algorithm like DPLL, is the operation of *undoing* or *erasing* one or more decision assignments (as well as the associated implications) from the current set of assignments A_{curr} with the aim of exploring unexplored partial assignments to the variables. Backtracking is performed when the CNF ϕ has been proved to be unsatisfiable in the sub-space under A_{curr}, *e.g.* by means of one or more conflicts.

In the DPLL algorithm (Algorithm 2.1) the backtracking performed is *chronological backtracking*. This has the following meaning. Say, x and y are two decision variables in A_{curr} such that $\lambda(x) < \lambda(y)$. Further, suppose x and y were currently being explored in their left branch assignment. Then the right branch assignment of y will always be explored before attempting the right branch assignment of x. Operationally, the DPLL algorithm, on encountering a conflict, would undo assignments from A_{curr} in reverse chronological order of decision level until it reached the *first* left branch assignment. At this point the assignment of this variable would be flipped and the search would resume.

Non-chronological backtracking (NCB) is based on the observation that it may be possible to establish that the sub-space under an unexplored right branch (*i.e.* the variable is currently being explored in its left branch in A_{curr}) cannot contain a solution for ϕ. Such a determination may be made by analyzing the current (and previous conflict conditions) and proving that the current conflict condition would continue to exist or repeat in this sub-space. Operationally, one way to do this[4] is to perform conflict analysis on the current conflict condition $\mathcal{X}$ and determine the lowest variable x_{lowest} (*i.e.* the one that has the decision level with the highest numerical value) in the set $A_{\mathcal{R}}(\mathcal{X})$. By the above argument the search can directly jump to $\lambda(x_{lowest})$ erasing all assignments between the current decision level (*i.e.* the level of the current conflict) and $\lambda(x_{lowest})$ and thereby bypassing or "jumping over" any explored right branches in these intermediate levels. In

[4]This is the method used by almost all SAT solvers implementing NCB.

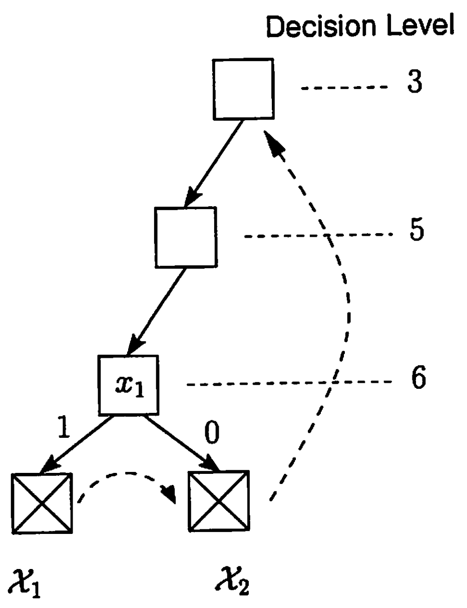


Figure 2.1: An Example of Non-chronological Backtracking [MSS99]

this sense the backtracking is *non-chronological.*

An instance of non-chronological backtracking can be seen on the CNF of Example 2.2. Figure 2.1 illustrates this. Consider conflict $\mathcal{X}_1$ from the example. Conflict analysis on $\mathcal{X}_1$ generates $A_{\mathcal{R}}(\mathcal{X}_1) = \{x_1 = 1, x_9 = 0, x_{10} = 0, x_{11} = 0\}$. From the above discussion on NCB we can deduce the backtrack level of the current conflict to be 6 *i.e.* the current level. Therefore, we simply flip the value of x_1 from 1 to 0. Note, that up to this point the backtracking is still chronological. However, the new assignment $x_1 = 0$ will immediately lead to another conflict $\mathcal{X}_2$ (this can be confirmed from the CNF given in Example 2.2). Analysis of this conflict yields

$$A_{\mathcal{R}}(\mathcal{X}_2) = \{x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2\}$$

From this the backtrack level can be deduced to be 3 which is indeed a non-chronological jump (the chronological backtrack level would have been 5).

2.7 Transformation of Non-Clausal SAT problems to CNF

For certain classes of Satisfiability problems the native representation is not conjunctive normal form. Since most SAT techniques and solvers are based on CNF representations, efficient procedures to convert instances from non-clausal forms to CNF are an essential component of SAT methodologies. In the following we discuss two such transformations which are especially relevant in the context of EDA applications.

2.7.1 CNF Representations of Propositional Formulas

Let V be a set of propositional variables. Well formed propositional formulas based on the set V are defined as follows [DW83, MS95]:

1. Any propositional variable $x \in V$ is a well formed formula.
2. If ρ is a well-formed formula, then so is $\overline{\rho}$.
3. If ρ and σ are well formed formulas then so are $(\rho \cdot \sigma)$, $(\rho + \sigma)$ and $(\rho \leftrightarrow \sigma)$.

From the above definition it is clear that CNF formulas are a special case of well formed propositional formulas. A simple procedure to convert an arbitrary well formed propositional formula to a CNF representation is as follows [DW83, MS95].

1. Express equivalence operations in terms of conjunction, disjunction and negation operations through the transformation: $(\rho \leftrightarrow \sigma) \equiv ((\overline{\rho} \cdot \overline{\sigma}) + (\rho \cdot \sigma))$.
2. Apply De Morgan's laws to expand out all negations, other than those associated with single variables.
3. Absorb double negations over single variables using the identity: $\overline{\overline{x}} \equiv x$.
4. Repeatedly apply the distributive law: $((l_1 \cdot l_2) + (l_3 \cdot l_4)) \equiv (l_1 + l_3) \cdot (l_1 + l_4) \cdot (l_2 + l_3) \cdot (l_2 + l_4)$ till a CNF is obtained.

Example 2.3 *The formula $\phi = \overline{(x \leftrightarrow y)} + \overline{(y \leftrightarrow z)}$ is a well formed propositional formula. It can be converted to a CNF representation using the above procedure. Applying Step 1 gives $\phi = \overline{(\overline{x}\overline{y} + xy)} + \overline{(\overline{y}\overline{z} + yz)}$ which after Steps 2 & 3 transforms to $(x+y)(\overline{x}+\overline{y})+(y+z)(\overline{y}+\overline{z})$. Finally, applying Step 4 and simplifying using basic axioms of Boolean Algebra[5], yields the CNF $\phi = (x+y+z)(\overline{x}+\overline{y}+\overline{z})$.*

Example 2.4 *Consider the propositional formula $\phi_1 = (\ldots(x_1 \leftrightarrow x_2)\ldots) \leftrightarrow x_N)$. It can be shown that applying the above procedure will yield a CNF with 2^{N-1} clauses.*

[5] $x + x = x$, $x \cdot x = x$, $1 \cdot x = x$ and $x + \overline{x} = 1$

This example illustrates that the above procedure to convert a well formed propositional formula to CNF can sometimes result in a CNF representation that is exponentially larger than the size of the original propositional formula. In Example 2.4 the size of the original ϕ is linear in N while the CNF is exponential in N. A transformation procedure proposed by G. Tseitin [Tse68] addresses this problem. Tseitin's transformation is defined as follows.

Definition 2.2 (Tseitin's Transformation [Tse68]) *Let ϕ be the given well formed propositional formula.*

1. *Associate a new propositional variable x_η with each sub-formula η contained in ϕ*[6]*, such that x_η and η always assume the same propositional value.*
2. *For each operation in ϕ, i.e. for each sub-formula of ϕ (including ϕ itself) create a set of clauses as follows:*
   - *If $\eta = \rho \cdot \sigma$ (where η, ρ and σ are all sub-formulas of ϕ) define clauses $(\overline{x_\eta} + x_\rho) \cdot (\overline{x_\eta} + x_\sigma) \cdot (x_\eta + \overline{x_\rho} + \overline{x_\sigma})$.*
   - *If $\eta = \rho + \sigma$ define clauses $(x_\eta + \overline{x_\rho}) \cdot (x_\eta + \overline{x_\sigma}) \cdot (\overline{x_\eta} + x_\rho + x_\sigma)$.*
   - *If $\eta = \rho \leftrightarrow \sigma$ define clauses $(\overline{x_\eta}+x_\rho+\overline{x_\sigma})\cdot(\overline{x_\eta}+\overline{x_\rho}+x_\sigma)\cdot(x_\eta+x_\rho+x_\sigma)\cdot(x_\eta+\overline{x_\rho}+\overline{x_\sigma})$.*
   - *If $\eta = \overline{\rho}$ define clauses $(\overline{x_\eta} + \overline{x_\rho}) \cdot (x_\eta + x_\rho)$.*
3. *Finally add a unit-literal clause (ϕ) to assert that the formula is required to be true.*
4. *The conjunction of all clauses generated above defines the desired CNF representation of ϕ, whose satisfiability we seek to determine.*

Consider the CNF ϕ_1 from Example 2.4. Let us apply Tseitin's transformation, described above, to this formula. ϕ_1 has $N-1$ sub-formulas, $\eta_1, \dots, \eta_{N-1}$ defined as $\eta_i = \eta_{i-1} \leftrightarrow x_{i+1}$ for $i = 2, \dots, N-1$ and $\eta_1 = x_1 \leftrightarrow x_2$. Hence, $N-1$ new propositional variables $x_{\eta_1}, \dots, x_{\eta_{N-1}}$ are created and for each η_i the clauses $(\overline{x_\eta} + x_{\eta_{i-1}} + \overline{x_{i+1}}) \cdot (\overline{x_\eta} + \overline{x_{\eta_{i-1}}} + x_{i+1}) \cdot (x_\eta + x_{\eta_{i-1}} + x_{i+1}) \cdot (x_\eta + \overline{x_{\eta_{i-1}}} + \overline{x_{i+1}})$ are added to the CNF. Thus, the resulting CNF has $4 \times (N-1)+1$ clauses and is linear in the size of the original propositional formula.

[6] ϕ is also a sub-formula of itself.

Tseitin's transformation is the most popular method for transforming general propositional formulas into CNF. However, other polynomial size transformations have also been developed. The interested reader is referred to [MS95] for more details on these methods.

2.7.2 Solving SAT problems on Logic Circuits

Many EDA applications using SAT models involve SAT problems posed on logic circuits[7]. Most of the work on SAT formulations of logic circuit EDA problems has focused on combinational logic circuits and such will be the focus of the research presented in this dissertation as well. Therefore, in the following discussion and in the rest of the dissertation the term *logic circuit* will be used to refer to a combinational logic circuit, unless explicitly noted otherwise.

Traditional solutions to EDA problems have sometimes employed algorithms that worked off the native representation of the logic circuit, often performing a branch-and-bound search similar to a DPLL-like algorithm, *e.g.* the **PODEM** algorithm for combinational ATPG [Goe81]. However, recent advancements in CNF based SAT solvers have prompted the formulation of such EDA problems in terms of CNF based SAT. In the following we review the essential elements of some popular techniques for transforming SAT problems on logic circuits to CNF.

For the purpose of solving SAT problems posed on combinational logic circuits, it is adequate to work with an abstract representation of a multi-level combinational logic circuit, known as a *Boolean network* [BRSVW87]. In this dissertation a given combinational logic circuit C is indistinguishable from its corresponding Boolean network.

A Boolean network is a directed acyclic graph (DAG), where a node without any incoming edge represents a *primary input* and a node without any outgoing edge represents a *primary output*. All other nodes represent intermediate gates[8]. A Boolean function is associated with each intermediate node. There is an edge from node n_i to a node n_j if the function associated with n_j explicitly depends on n_i. If there is an edge from node n_1 to node n_2, n_1 is said to be a *fanin* node of n_2 and n_2 is said to be a *fanout* node of n_1. If there is a directed path from n_1 to n_2 in the Boolean network, n_1 is said to be in the *transitive fanin* of n_2 and n_2 is said to be in the transitive fanout of n_1.

[7]An exception are SAT formulations of physical design problems in EDA *e.g.* [WR98].

[8]It is assumed that primary inputs and outputs do not compute any non-trivial function.

Further, every node in the Boolean network (including primary inputs, primary outputs and the outputs of internal gates) is associated with a propositional variable. For a node n whose output variable is y, $I(y)$ denotes the fanin of the node (*i.e.* the input variables of the gate) and $f_y(I(y))$ denotes the Boolean function implemented by the gate.

Most SAT problems arising from combinational logic circuits, such as combinational ATPG, combinational equivalence checking *etc.* can be posed as a generic satisfiability problem on a *suitable* circuit. This generic satisfiability problem is known as *CIRCUIT-SAT*. In the following we define this problem on a circuit with a single primary output and present methods to transform such a problem into a CNF based SAT problem. While a discussion describing the transformation of every known logic circuit based SAT problem into CNF is beyond the scope of this discussion, the transformation methods for *CIRCUIT-SAT* apply to most such problems and provide a good starting point for the remaining problems.

Definition 2.3 (CIRCUIT-SAT) *Given a single output Boolean circuit C, the circuit satisfiability problem on C, denoted as CIRCUIT-SAT(C), seeks to determine a logic value assignment (partial or complete) to the primary inputs of C under which the primary output of C evaluates to* 1. *Such an assignment, if one exists, is called a satisfying assignment of C, otherwise the instance CIRCUIT-SAT(C) is said to be unsatisfiable.*

Given a CIRCUIT-SAT problem on a logic circuit C one naive method to transform this into a CNF SAT problem would be the following. First construct a single propositional formula for the function of the primary output of C by starting with the primary output (say y) and recursively substituting $f_w(I(w))$ into $f_y(I(y))$ for each $w \in I(y)$. Then this propositional formula can be transformed into CNF using one of the methods outlined in Section 2.7.1. The problem with this approach is that the propositional formula obtained can be exponentially larger than the circuit representation C, thus making the approach impractical.

What is used in practice is essentially an adaptation of Tseitin's transformation to combinational circuits. For each gate in C, with output y, we construct a *consistency function*, ξ_y [MS95] defined as follows.

$$\xi_y(I(y), y) = \overline{[g(I(y)) \oplus y]} \tag{2.6}$$

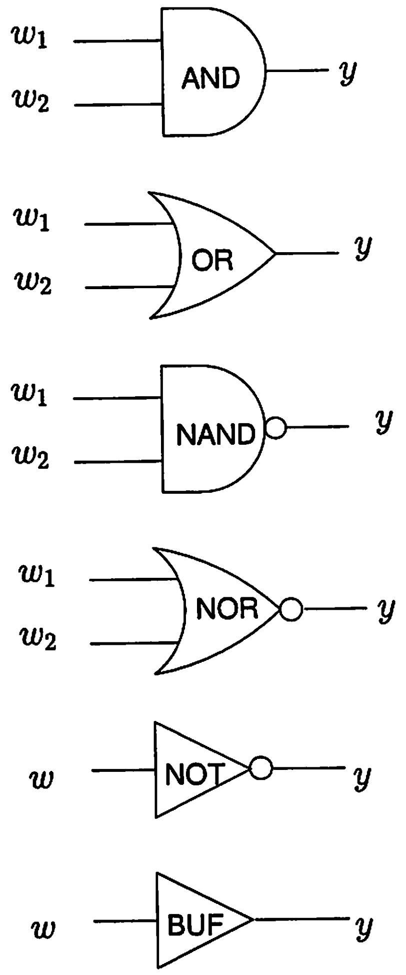
Gate type	Gate Function	ξ_y
AND	$y = AND(w_1, \ldots, w_k)$	$\left(\prod_{i=1}^{k}(w_i + \overline{y})\right) \cdot \left(\sum_{i=1}^{k}\overline{w_i} + y\right)$
OR	$y = OR(w_1, \ldots, w_k)$	$\left(\prod_{i=1}^{k}(\overline{w_i} + y)\right) \cdot \left(\sum_{i=1}^{k} w_i + \overline{y}\right)$
NAND	$y = NAND(w_1, \ldots, w_k)$	$\left(\prod_{i=1}^{k}(w_i + y)\right) \cdot \left(\sum_{i=1}^{k}\overline{w_i} + \overline{y}\right)$
NOR	$y = NOR(w_1, \ldots, w_k)$	$\left(\prod_{i=1}^{k}(\overline{w_i} + \overline{y})\right) \cdot \left(\sum_{i=1}^{k} w_i + y\right)$
NOT	$y = NOT(w)$	$(w + y) \cdot (\overline{w} + \overline{y})$
BUFFER	$y = BUFFER(w)$	$(w + \overline{y}) \cdot (\overline{w} + y)$

Table 2.3: CNF formulas for simple gates

The Boolean function $\xi_y(I(y), y)$ evaluates to 1 when the values of the inputs of $I(y)$ are consistent with the value of the output y as per the Boolean function f_y associated with the node. To construct the CNF for *CIRCUIT-SAT(C)* we construct the CNF representation for the consistency function of each node in C, take the conjunction of all these CNFs and add an additional unit-literal clause asserting the primary output of C to 1. The resulting representation is the CNF SAT formula corresponding to the instance *CIRCUIT-SAT(C)*.

The above procedure requires the construction of the CNF for $\xi_y(I(y), y)$. There are several ways to do this including obtaining a *product-of-sums (POS)* representation by directly simplifying the truth table of $\xi_y(I(y), y)$ [MS95] or by using one of the two procedures described in Section 2.7.1. However, the most popular method is to first decompose each gate in the original circuit C into a set of simple gates (AND, OR, NOT, NAND, NOR *etc.*) by using one of several well-known Boolean function decomposition techniques [BRSVW87]. Then the above procedure is applied on this circuit of simple gates by using rules such as those given in Table 2.3 to construct the CNF for $\xi_y(I(y), y)$ of each simple gate. Figure 2.2 tabulates the CNF formulas for 2-input and 1-input simple gates. These are just the formulas given in Table 2.3 evaluated for the case $k = 2$.

The reader will note that this procedure is identical to Tseitin's Transformation for propositional formulas (Definition 2.2). Here the addition of extra variables is obviated



$$(w_1 + \overline{y})(w_2 + \overline{y})(\overline{w_1} + \overline{w_2} + y)$$

$$(\overline{w_1} + y)(\overline{w_2} + y)(w_1 + w_2 + \overline{y})$$

$$(w_1 + y)(w_2 + y)(\overline{w_1} + \overline{w_2} + \overline{y})$$

$$(\overline{w_1} + \overline{y})(\overline{w_2} + \overline{y})(w_1 + w_2 + y)$$

$$(w + y)(\overline{w} + \overline{y})$$

$$(w + \overline{y})(\overline{w} + y)$$

Figure 2.2: CNF formulas for 2-input & 1-input simple gates

by the existence of intermediate gate output variables which do indeed define sub-formulas of the overall circuit formula. Further, the formulas in Table 2.3 are simple generalizations of the rules in Step 2 of Definition 2.2.

Chapter 3

SAT Algorithms in EDA: Recent Developments

Recent years have seen dramatic improvements in the performance of algorithms and tools for SAT, allowing much larger problem instances to be solved and greatly expanding the realm of applicability of SAT solvers. Spurred by the recent advancements in SAT solver technology, SAT algorithms have been successfully applied to problems from a wide variety of EDA applications [MSS00]. In fact, SAT solvers are expected to have an impact on EDA applications similar to what BDDs have had since their introduction more than a decade ago.

This chapter surveys the recent developments in the area of SAT algorithms and their application to EDA problems. Some of the early progress in this domain is comprehensively described in two other works [GPFW97, MSS00]. Progress in this field can be categorized along two axes, namely 1.) improvements to core SAT algorithms and techniques and 2.) application of SAT algorithms to various EDA problems. These are surveyed in Section 3.1 and Section 3.2 respectively.

Traditionally, the propositional satisfiability problem is posed on a *Conjunctive Normal Form (CNF)* formula. The SAT problem can be quite naturally posed on other representations of Boolean formulas *e.g.* DNF, multi-level logic circuits, *etc.* Such works are surveyed in Section 3.1. However, unless otherwise stated, the discussion is with respect to a CNF representation.

3.1 SAT Algorithms & Tools

SAT algorithms can be broadly divided into two categories.

1. **Complete SAT algorithms:** Given a SAT problem, a complete SAT algorithm will either find a satisfying assignment for the problem or prove that no such assignment exists. Such algorithms are mostly deterministic[1] in nature and involve some kind of organized exploration of the complete solution space.

2. **Incomplete SAT algorithms:** For a given SAT instance such an algorithm either 1.) returns with a satisfying assignment for the problem or 2.) terminates without an answer. The idea behind this class of algorithms is to spend a given amount of resources in "quickly" finding solutions to a large fraction of satisfiable instances and to leave the "hard-to-satisfy" and unsatisfiable instances to the complete (albeit expensive) SAT algorithms. These algorithms are partly stochastic in nature and usually involve some kind of sampling of the solution space (in contrast to an organized exploration).

3.1.1 Complete SAT Solvers

3.1.1.1 SAT Solvers based on Propositional Formulas

The two earliest complete methods for the SAT problem are the *Davis-Putnam (DP)* method [DP60] and the *Davis-Putnam-Logemann-Loveland (DPLL)* procedure [DLL62], reviewed in Section 2.5 and Section 2.4 respectively. The former employs a proof-theoretic approach based on an organized application of the *clause resolution* operation. The latter does a branch-and-bound exploration of the space of all possible Boolean assignments to the variables of the CNF formula. Most current SAT solvers are derivatives of the DPLL procedure augmented with a limited amount of resolution based reasoning.

The **GRASP** SAT solver [MSS99] was the first to introduce the techniques of *non-chronological backtracking* and *conflict driven learning* (these have been reviewed in Section 2.6) into DPLL based SAT solvers. These techniques are founded on the notion of *conflict analysis* which seeks to determine and analyze the causes of a given conflict encountered during the search. The above techniques were independently developed and proposed in the **RELSAT** SAT solver [BS97] by Bayardo & Schrag. A novel feature of

[1]Probabilistic SAT methods can also be made to be complete.

RELSAT is the notion of *relevance based learning* whereby a recorded clause is discarded when at most i literals in it have changed value since its recording. This feature is used to control the growth of the CNF formula as is the notion of recording only clauses bounded by a certain size (*size bounded learning*).

SATO [ZS96] proposed a new data-structure for performing efficient Boolean Constraint Propagation (BCP), since this operation is a significant component of the computation time of all DPLL based solvers. **SATO** was later extended [Zha97] to incorporate the techniques proposed in **GRASP**. More recently, Moskewicz *et al.* have proposed the **Chaff** SAT solver [MMZ+01]. **Chaff** is a very efficient and optimized implementation combining the data-structure of **SATO**, the search pruning techniques of **GRASP**, the notion of relevance based learning from *RELSAT* and the technique of *search restarts* from [GSK98] with some of its own enhancements. The solver employs a novel cheap and efficient decision heuristic which significantly speeds up the computation, as well as garbage collection techniques to efficiently manage the clause database. This solver is arguably the current state of the art in terms of implementation and algorithmic advancements in SAT solvers.

Another class of techniques is based on a patented method [Stå] by Gunnar Stålmarck [SS00]. The method is in use in a commercial tool offering by Prover Technologies [pro] and has had some success in the verification domain [Bor97]. Stålmarck's method was originally proposed to work off a non-CNF representation but has been adapted to work on CNF representations in the **HeerHugo** solver [GW99]. HeerHugo also employs some restricted, rule-based application of resolution.

An interesting set of contributions orthogonal to progress in SAT solvers per se is in the area of *incremental satisfiability* [KMSS00b, WKS01]. The motivation behind this research is that in many practical applications such as delay-fault testing, timing analysis and bounded model checking *etc.* the task is to solve a set of SAT instances which share a large percentage of common clauses. Usually these instances are merely different "questions" posed on the same basic physical system. Incremental satisfiability techniques attempt to efficiently solve such a set of related SAT instances by using specific information from the solution process of one instance to aid the solution of other instances. The **SATIRE** [WKS01] solver which is built on top of **GRASP** supports this incremental feature and also supports some forms of non-Boolean constraints in the input SAT instance.

3.1.1.2 Circuit Based Boolean Reasoning

A multi-level logic circuit can also be viewed as a representation of the function(s) realized by the primary output(s) of the circuit. Thus, SAT problems can be posed and solved on circuit representations as well. The combinational ATPG problem can be quite naturally posed as a SAT problem. Traditionally, ATPG tools *viz.* **PODEM** [Goe81], **SOCRATES** [STS88] and [ZRP97] have performed Boolean Reasoning on the original circuit representation. *Recursive Learning* [KP94] proposed by Kunz and Pradhan is a complete technique to perform Boolean reasoning on multi-level circuits[2]. While recursive learning has not proved to be very effective as a stand-alone technique, restricted forms of it have been used in SAT solvers [MSG99, KGP01]. In fact, Stalmårck's method and its CNF implementation in **HeerHugo** also bear some similarity to recursive learning. Recently, Kuehlmann *et al.*[KGP01] have proposed a Boolean reasoning engine that operates directly on a circuit graph representation. It includes a state of the art branch and bound SAT solver that incorporates features of advanced CNF solvers as well as a number of other, circuit specific, search pruning techniques.

SAT problems posed on logic circuits are frequently solved by transforming the problem to CNF, albeit at the cost of hiding structural information that a circuit-based technique could exploit. In some applications such information can be invaluable in solving the problem. The early works using CNF based SAT for circuit problems [Lar92, SBSV96] performed a single pass of static learning to extract important functional information from the circuit structure, which was added as clauses to the CNF database. Silva *et al.* [eSSMS99] have proposed adding a layer on top of a traditional CNF Solver (**GRASP** in this case) which passes structural information from the circuit to aid the solver. Tafertshofer *et al.* [TGH97] have proposed a new specialized data-structure called an *implication graph* which has the topology to represent the circuit structure but is general enough to homogeneously integrate some forms of learning, traditionally supported by CNF solvers. [GF01] proposes a variant of this data-structure and some novel static and dynamic learning techniques that can be applied to it.

[2]In other words solve SAT problems on circuits.

3.1.1.3 Other improvements to SAT solvers

A recent direction in SAT solving has been the use of dedicated reconfigurable hardware architectures [ZMAM99, AdS00, dSMSA01]. The objective here is to exploit the fine-grained parallelism of hardware to solve the problem faster. Unfortunately, given today's technology, the overhead associated with mapping a SAT instance to hardware is too much to justify the use of this approach, except for very large problem instances.

Some other recent developments have been aimed at enhancing specific aspects of a typical complete SAT algorithm. Variable ordering (decision heuristic) issues have been discussed in [LA97] and [AMS01]. The **SATZ** solver [LA97] uses a partial one-step lookahead scheme and chooses the variable giving the greatest number of BCP assignments[3] as the next branching variable. [AMS01] has proposed a novel static variable ordering scheme based on mincut partitioning.

Rule-based learning or CNF simplification approaches constitute another class of enhancements that have been proposed. The *equivalency reasoning* approach proposed by Li [Li00] uses a set of rules to recognize and deduce new 2 and 3-variable equivalency conditions (biconditionals) using unit-literal propagation and pattern recognition on the CNF. This is done at each step of the DPLL procedure. Interestingly, **SATZ** enhanced with equivalency reasoning (**EqSatz**) is able to solve almost all the BMC examples [BCC+99] in times comparable to or faster than **Chaff**. Le Berre [Ber01] proposed a similar approach where a set of rules is used to deduce special cases of implications, equivalences and biconditionals by examining consequences of both assignments to a variable with full BCP (here the procedure has commonalities with level-1 recursive learning). Marques-Silva [MS00] proposes a rule-based pattern-matching approach to simply the CNF in a preprocessing step. His rules detect equivalent variables and simple 2-literal clauses deduced through special cases of resolution. While each of the above rule-based approaches have claimed limited success in their respective experimental environments, they have not been proven or incorporated in a leading edge SAT solver such as **Chaff**. An orthogonal category of research is aimed at solving SAT on decision diagram data-structures rather than CNFs or logic circuits; [AMS01] use ZBDDs while Williams *et al.* [WAH01] propose using *Boolean Expression Diagrams (BEDs)*.

There is also a body of work aimed at trying to make complete SAT algorithms

[3]As opposed to computing some estimate of the potential BCP.

partly stochastic in nature. The most notable contributions in this direction are the use of *randomization and restarts* and *dynamic backtracking*. The notion of search restarts is used to partially remedy "early bad" variable choices made by the decision heuristic. Simply put, the strategy is to abort the search after a certain number of backtracks and restart from scratch, retaining all or some of the clauses that were learned in the search thus far. Randomization is used mainly in the decision heuristic, say by randomly choosing the next branching variable from a set of "good" candidates. These techniques were introduced in [GSK98], in the context of AI applications and incorporated in the **GRASP**[BMS00] and **Chaff** [MMZ+01] solvers. The general consensus regarding their efficacy is that they do not have an appreciable overhead in the instances where they are not effective but give significant speedups (up to one order of magnitude) in cases where they are useful. Dynamic backtracking was proposed in the context of incomplete solvers in [Gin93] and recently adapted for complete search [LBMS01]. The basic idea is that on a conflict, instead of undoing the last variable responsible for the conflict, the algorithm randomly undoes any one of the responsible assignments. This and other variants of dynamic backtracking schemes are still very much in the experimental stage. These techniques will need to mature before a definitive statement on their utility can be made.

3.1.2 Incomplete SAT Solvers

Incomplete SAT solvers shot to prominence with the work on **GSAT** [SLM92] where greedy local search was shown to outperform state of the art complete algorithms, for some applications from the Artificial Intelligence domain. The **GSAT** algorithm consists of a set of *tries*. In each try, starting from an random initial assignment, variables are greedily chosen and *flipped* with the cost function of maximizing the number of satisfied clauses. This is done till all the clauses are satisfied or the algorithm exhausts a predetermined maximum number of tries. The **GSAT** algorithm was later developed into **WSAT** *(Walksat)* [SKC96] by adding a small amount of noise to escape local minima. The algorithm is again organized as a sequence of *tries*, where each try is a sequence of *flips*. Each flip is made by first randomly picking an unsatisfied clause and picking (either at random or according to a greedy heuristic) a variable within that clause to flip. In the experience of the EDA community the **GSAT** and **WSAT** algorithms per se are not well suited to EDA applications because EDA SAT problems represent highly constrained spaces. Even successors to **WSAT** like [KMS97], which are designed to operate in constrained spaces, have not found acceptance in EDA. Recent work on stochastic local search employing dynamic backtracking [Gin93, Pre00] seems promising but is yet to be extensively tested on EDA problems.

However, incomplete stochastic techniques such as random and weighted random simulation are routinely used in ATPG and verification applications. Such techniques are in some sense similar to algorithms like **WSAT**. Also, a simple variant of **WSAT** adapted to work off BDDs was successfully used by Singhal & Burch in their equivalence checker [BS98]. Therefore, it may be just a matter of time before effective incomplete solvers for EDA applications are developed.

3.2 SAT Applications in EDA

Several EDA applications have problems that can be quite naturally formulated in terms of SAT which is rapidly becoming the method of choice for solving an increasing fraction of these problems. Verification has provided the richest application domain for use of SAT in EDA. However, applications in other areas exist as well. These are reviewed in Sections 3.2.1 and 3.2.2 respectively. For a more complete list of some of the early applications of SAT in EDA refer to [GPFW97].

3.2.1 SAT in Verification

Combinational ATPG was one of the earliest applications of SAT in EDA. Efficient SAT-based Combinational ATPG tools such as **NEMESIS** [Lar92], **TEGUS** [SBSV96] and **TIP** [TG99] were developed for the ***single stuck-at*** and ***bridging*** fault models. SAT models have also been used for delay fault testing [CG96, KMSS00a]. The formulation of SAT based combinational ATPG for single stuck-at faults, originally proposed by Larrabee [Lar92], and later used by all subsequent works is briefly reviewed in Section 4.1. A recent, related application has been in using a hybrid of 3-SAT and linear programming in functional vector generation for HDL models [FDK98].

Combinational Verification (CEC) has also provided a fertile ground for application of SAT methods. The **HANNIBAL** [Kun93] tool employs recursive learning on the circuit supplemented with an ATPG engine, [TGH97] perform Boolean reasoning on their *implication graph* data-structure, [MSG99] proposed using the GRASP solver preceded by a pre-processing phase of CNF-based recursive learning. However, state-of-the art CEC tools use a cut-point based approach where SAT is used in combination with a number of other engines such as BDDs, simulation and structural analysis algorithms (*e.g.* graph isomorphism) [GA98, BS98, MJT$^+$99, PK00]. [BS98] uses a BDD and simulation framework complemented with randomized SAT algorithms modified to work off the BDD data-structure. [MJT$^+$99] uses a filter-based approach where a single cut-point check passes through a sequence of engines (such as simulation, structural approaches, SAT, BDDs) of increasing power and complexity. In [PK00] Paruthi & Kuehlmann use an interleaved invocation of BDDs and a SAT solver, with increasing thresholds, to accomplish each equivalence check.

A relatively recent application is the use of SAT for verifying safety properties on sequential systems. One such method that has become popular is *Bounded Model Checking* [BCCZ99, BCC$^+$99]. Simply put, it involves unrolling a sequential circuit for a specified number of time-frames and constructing a CNF which asserts that a particular safety property is violated on the unrolled circuit. A case study for the application of this method is presented in [BCRZ99] and improvements to the original formulation are reported in [Sht00, Sht01]. The proposed improvements include variable ordering issues and ideas drawn from incremental satisfiability.

Bounded model checking techniques are incomplete verification techniques in that they reason only about the state-space covered by a fixed number of time-frames. A fairly recent direction in SAT research is to explore the use of SAT methods in a conventional (*i.e.* complete) model checking framework for problem such as reachability analysis and image computation [SS90, SSS00, ABE00, BC00, WBCG00, GYAG00, BLM01]. Abdulla *et al.* [ABE00] use a non-canonical data-structure called a *Reduced Boolean Circuit (RBC)* to represent the functions (*i.e.* the transition relation, state sets *etc.*) and a conventional SAT checker (the **PROVER** tool based on Ståmarck's method) is used for various SAT checks required during the entire process *viz.* checking for the fixed-point. The quantification step of image computation is implemented on the RBC itself using quantification axioms such as *inlining* and *scope reduction* to alleviate size explosion of the RBC. [WBCG00] use a similar approach where *Boolean Expression Diagrams (BED)* are used in place of the RBC, **SATO** is used as the SAT checker and some additional pruning techniques are employed to control size explosion of BEDs. [SSS00] and [BC00] report results on using powerful variants of *induction* and a SAT solver (**PROVER** in this case) to perform property checking. Gupta *et al.* [GYAG00] have proposed an approach of combining SAT and BDDs to perform image computation. BDDs are used to represent the state sets and a CNF to represent the transition relation. The next-state computation is done through a combination of BDDs and SAT algorithms. Several improvements to this basic framework are proposed in [GGYA01] and [GYA$^+$01b].

Velev *et al.* investigated the use of SAT procedures in microprocessor verification. In [BGV01] the microprocessor verification task is expressed in the logic of equality with uninterpreted functions and then efficiently reduced to propositional satisfiability (SAT). In a recent series of works by the same authors this formulation has been extended to model superscalar microprocessors [VB99], superscalar processors with multi-cycle functional units, exceptions and branch prediction [VB00] and VLIW processors with speculative execution [Vel00].

A verification methodology, gaining popularity in the EDA industry, is a class of methods known as semi-formal techniques. The objective is to use an efficient combination of simulation and formal techniques to do a more effective validation of the design, find more bugs or do a limited amount of formal checking. The **SIVA** tool [GYA$^+$01a] and the **Ketchum** tool [HSH$^+$00] are two such works that use SAT methods as one of the engines in combination with BDDs, simulation, symbolic simulation and structural ATPG methods.

3.2.2 Other Applications

In the area of synthesis, SAT algorithms have been used for *exact timing analysis* [MSS$^+$91, eSMSSS97] and logic optimization through *redundancy removal* [EC95]. More recently, SAT models have been applied to *Crosstalk noise analysis* [CK99] and logic optimization using *don't cares* [SB01].

Physical design has not seen many applications of SAT, its only notable use being in FPGA Routing [WR98, NSR99, NASR01]. The reason for this is that many problems in physical design are geometrical in nature and a propositional encoding of such problems usually produces very large SAT instances, which cannot be solved by current SAT methods. Other physical design problems are inherently non-discrete and thereby best suited for continuous optimization methods.

3.3 Conclusions

The last few years have seen the growth of SAT solvers in EDA from being an algorithmic problem of academic interest to a powerful reasoning tool and an enabling technology for several applications. A host of SAT solvers and extensive suites of SAT benchmarks are now available in the public domain (see **SATLIVE!** [Ber], **SATLIB** [Ber] and **Sat-Ex** [SC01]) to facilitate research in SAT algorithms and applications. Realizing the commercial potential of SAT solvers, several companies, *e.g.* **Prover Technologies** [pro] and **Greentech Computing Inc.** [gre] are commercially marketing SAT solvers and services associated with their use in various applications. However, despite the dramatic progress in this area, only a small fraction of the immense potential of SAT for EDA has been realized. The next few years promises to be an exciting time for EDA professionals engaged in SAT research.

Chapter 4

The Practical Complexity of SAT Based ATPG

This chapter presents an analysis of the complexity of an important and well studied EDA application that uses SAT methods. The underlying objective is to make such an analysis more realistic by accounting for salient characteristics of problem instances encountered in real life. The problem examined is the *combinational automatic test pattern generation (ATPG)* problem for the *single stuck-at fault* model.

Combinational ATPG techniques find widespread use in a number of EDA applications. In addition to generating test patterns for testing digital combinational circuits, for which they were originally proposed, they have proved to be effective tools of logic optimization [DMSV88, EC95] and have recently found application in verification techniques as well [Bra93]. The analysis presented in the sequel also sheds light on the following paradox regarding the combinational ATPG problem.

It has been known for more than two decades that the combinational ATPG problem is NP-complete [IS75]. This means that unless $P = NP$, there cannot exist an algorithm which solves an arbitrary instance of this problem in polynomial time. However, as early as 1979, Williams and Parker [WP79] claimed that for practically encountered instances of the problem the complexity of combinational ATPG is only $O(n^3)$. In fact, the widespread use of ATPG-based techniques can largely be attributed to the relative ease with which large instances of the problem are solved in practice.

We corroborated the claim that combinational ATPG is easily solvable in prac-

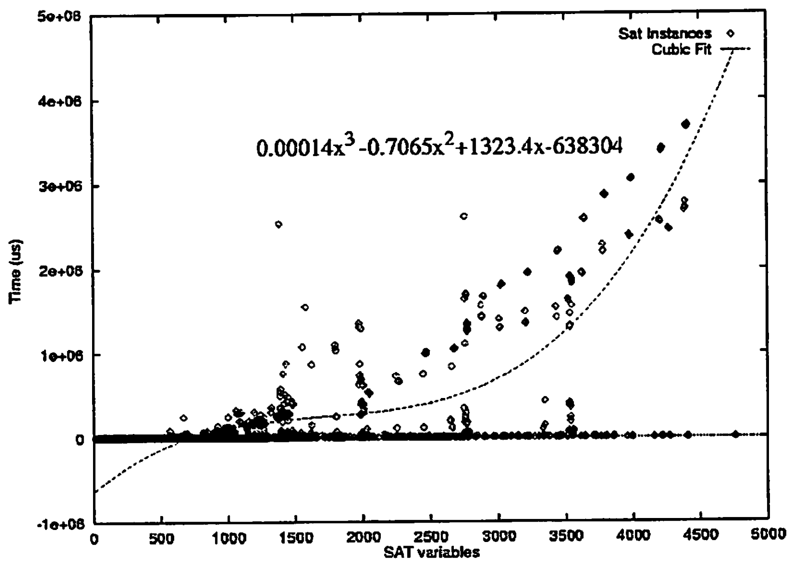


Figure 4.1: Results of TEGUS on ATPG-SAT instances

tice by performing the following experiment. ATPG was carried out on the combinational circuits from the **MCNC91** [Yan91] and **ISCAS85** [BF85] benchmark suites, using **TEGUS** [SBSV96], an ATPG tool based on a Boolean satisfiability (SAT) formulation. The time to solve each SAT instance was recorded as a function of the size of the instance (number of SAT variables) and plotted in Figure 4.1. Of the 11,000 SAT instances generated, some with over 15,000 variables, over 90% were solved in less than 1/100th of second; these were removed from the plot for clarity. The remaining instances exhibited roughly a cubic growth in execution time. Thus, the theoretical worst case complexity of ATPG, *i.e.* the fact that it is NP-complete, would seem to be a poor indicator of the practical ease of the problem. The work in this chapter is one of the first attempts to offer a theoretical explanation for the practical ease of ATPG. In the rest of the chapter, the term ATPG is used to refer to combinational ATPG.

The practical ease of ATPG suggests that there is some underlying property common to real-life ATPG instances which makes them tractable. These instances are usually derived from practical VLSI circuits. Therefore, we develop a characterization of the complexity of solving ATPG in terms of a topological circuit property, namely *cut-width*. We also demonstrate, through theoretical arguments and experiments on practical circuits, that a large class of interesting circuits have small *cut-widths*, provably permitting efficient ATPG.

We use a popular formulation based on SAT as our working model of the ATPG algorithm. This formulation is based on Tseitin's transformation for transforming a CIRCUIT-SAT problem into an instance of CNF Satisfiability (reviewed in Section 2.7.2). The formulation was originally proposed by Larrabee [Lar92] and later developed by Stephan et al. [SBSV96]. It must be noted that although the current analysis is intended for the ATPG problem, the same basic analysis framework could be applied to any EDA problem that uses a CIRCUIT-SAT based formulation.

The rest of the chapter is organized as follows. We begin with some definitions and notation in Section 4.1. In Section 4.2 we briefly discuss some seemingly promising approaches for analyzing the complexity of ATPG instances, based on existing results and analysis techniques. We argue that these approaches provide only an incomplete or inconclusive answer to the practical complexity of ATPG. Section 4.3 presents our model of the backtracking based algorithm for solving SAT, the *cut-width* property of circuits, and an analysis of the complexity of ATPG in terms of cut-width. In Section 4.4 we present both theoretical arguments and empirical results to show that a cut-width based argument does in fact predict a polynomial runtime of ATPG on a large class of practical circuits. In Section 4.5 we present interesting parallels and points of contrast between our results and published work addressing bounds on the size of binary decision diagrams (BDDs). A summary and discussion of the salient results presented in this chapter is given in Section 4.6.

4.1 Definitions and Notation

Definition 4.1 (Single Stuck-at-fault) *Given a Boolean network C [BRSVW87], a single stuck-at fault $\psi = \psi(x, \mathcal{B})$ is one which causes a net x in C to be permanently stuck at logic value $\mathcal{B}$ (where $\mathcal{B} \in \{0,1\}$).*

In the above, we consider a net to be the output of a gate (node) and all its associated fanout stems. Normally, one would distinguish the potential faults for different fanouts of a single net. However, for the purpose of our analysis we consider just one fault per net. This does not affect the generality of our results.

Definition 4.2 (Faulted Circuit) *Given a circuit C and a single-stuck at fault $\psi(x, \mathcal{B})$, the faulted circuit C_ψ is the original circuit C with the fault ψ operative i.e. the fault-net x asserted to $\mathcal{B}$.*

Definition 4.3 (The ATPG Problem) *Given a Boolean network C and a single stuck-at fault ψ, the ATPG problem $ATPG(C, \psi(x, \mathcal{B}))$ seeks to determine an assignment of Boolean values to the primary inputs of C (and thus C_ψ) such that fault net x has complementary logic values in C and C_ψ and at least one pair of corresponding primary outputs of C and C_ψ have complementary logic values. Such a Boolean assignment is said to be a test for the fault ψ. If no such assignment exists the fault is said to be untestable.*

The CIRCUIT-SAT problem, introduced earlier in Section 2.7.2 (Definition 2.3) can be easily generalized to multi-output circuits as follows.

Definition 4.4 (CIRCUIT-SAT) *Given a multi-output Boolean circuit C, the circuit satisfiability problem on C, denoted as CIRCUIT-SAT(C) seeks to determine a logic value assignment (partial or complete) to the primary inputs of C under which at least one of the primary outputs of C evaluates to* 1. *Such an assignment is called a satisfying assignment of C; if none exists the instance CIRCUIT-SAT(C) is said to be unsatisfiable.*

To simplify the discussion we use the above definition of CIRCUIT-SAT in the rest of the chapter. The transformation of a CIRCUIT-SAT problem posed on a multi-output circuit to a CNF Satisfiability problem proceeds on the same lines as discussed in Section 2.7.2 for a single output circuit. The only difference is that the unit-literal clause asserting the primary output to 1 is replaced by a clause which is the disjunction of all primary output variables. This enforces the requirement that at least one primary output must be set to 1. In the following treatment we will make no distinction between the *CIRCUIT-SAT* problem on a circuit C and the Boolean satisfiability (SAT) problem on its corresponding CNF formula $\phi(C)$. The set of variables of $\phi(C)$ will be denoted by V_C.

The ATPG problem can be naturally cast as a satisfiability problem by formulating it as a CIRCUIT-SAT problem on a suitable circuit, C_ψ^{ATPG} derived from the original circuit C and the fault ψ as follows [Lar92].

- $C^{sub,\psi}$: The sub-circuit of C containing all gates, inputs and outputs in the transitive fanin of the transitive fanout of the fault-point x.
- C_ψ^{fanout}: The sub-circuit of C_ψ corresponding to the *transitive-fanout* of x in C_ψ. The set of primary inputs of C_ψ^{fanout} comprise the *fault-boundary* of C (Figure 4.2). The inputs of C_ψ^{fanout} are fed from appropriate signal points in $C^{sub,\psi}$.

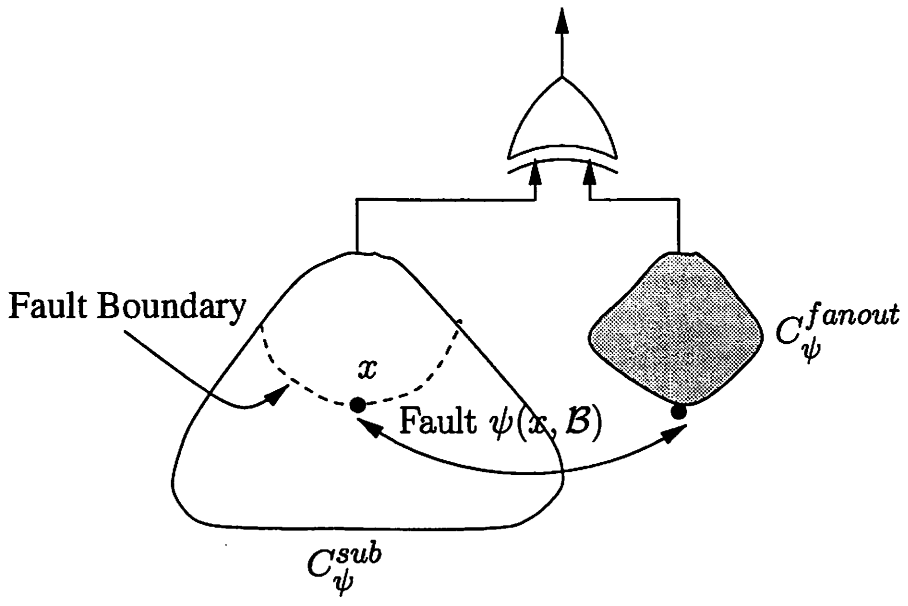


Figure 4.2: Circuit C_ψ^{ATPG} used for ATPG-SAT

- C_ψ^{ATPG}: The circuit obtained by the pairwise *XOR* of corresponding outputs of $C^{sub,\psi}$ and C_ψ^{fanout}.

The set of all satisfying assignments for the *CIRCUIT-SAT* instance C_ψ^{ATPG} gives precisely the set of all input vectors that test the fault ψ . Thus the ATPG problem *ATPG*$(C, \psi(x, \mathcal{B}))$ can be formulated as an instance of Boolean satisfiability denoted by *CIRCUIT-SAT*(C_ψ^{ATPG}). Henceforth, we will refer to this special instance of SAT as *ATPG-SAT*.

Definition 4.5 (ATPG-SAT problem) *ATPG-SAT refers to the SAT instance corresponding to an ATPG problem. Specifically, ATPG-SAT(C, ψ) refers to the SAT formula for testing the single stuck-at fault ψ on circuit C.*

Throughout this discussion, we assume that the circuits we deal with have gates with the number of fanins and number of fanouts bounded by k_{fi} and k_{fo}, respectively. We also assume the circuits are mapped to simple AND and OR gates, allowing inversions. The former restriction is enforced for practicality; design and technology constraints prohibit unlimited fanin and fanout. The latter restriction is enforced to facilitate the construction of the corresponding SAT formulas; it is difficult in practice to derive SAT formulas for arbitrary gates. TEGUS [SBSV96] enforces this latter condition for exactly this reason.

4.2 Applying Existing Techniques

In this section, we analyze three possible approaches based on an application of existing results and analysis techniques. We argue that none is capable of offering a conclusive or sufficiently general explanation for the practical complexity of ATPG.

4.2.1 Simple SAT Classes

Certain classes of SAT problems are known to be solvable in polynomial time. 2-SAT and Horn-SAT [GPFW97] formulas are two examples. If we could show that an interestingly large class of ATPG-SAT instances fall into one of the known polynomial time solvable SAT classes it would imply that the corresponding class of ATPG problems are efficiently solvable. We argue that this is highly unlikely, using the following reasoning.

Definition 4.6 (Complexity Index [BCHS94]) *Given a CNF Boolean formula ϕ, defined on Boolean variables $x_1, x_2, \dots x_n$ and having clauses $\omega_1, \omega_2, \dots, \omega_m$. The* complexity index *of ϕ is the optimal value $Z(\phi)$ of the following linear programming problem, $LP(\phi)$*

$$Z(\phi) = \min Z$$

$$\textit{such that}$$

$$\sum_{i \in P_k} \alpha_i + \sum_{i \in N_k} (1 - \alpha_i) \leq Z \quad (k = 1 \dots m) \textit{ and}$$

$$0 \leq \alpha_i \leq 1 \quad (i = 1 \dots n)$$

where P_k (N_k) is the set of positive (negative) literals in clause ω_k, and $\alpha_1, \alpha_2, \dots, \alpha_n$ are the variables of the LP problem, one each corresponding to variables $x_1, x_2, \dots x_n$ of the formula ϕ.

Boros *et al.* [BCH90] identified a fairly general class of efficiently (polynomial time) solvable SAT formulas known as *q-Horn* formulas. The set of q-Horn problems include several efficiently solvable classes of SAT formulas such as Horn-SAT, 2-SAT, Hidden-Horn-SAT, Extended-Horn-SAT *etc.*

Theorem 4.1 [BCH90] *q-Horn formulas have a complexity index of at most* 1.

Now, consider the circuit C_{sub} shown in Figure 4.3. Let ϕ_{sub} denote the CNF formula comprising the conjunction of the CNFs for the consistency functions of the gates

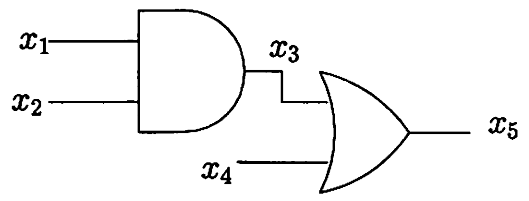


Figure 4.3: A simple sub-circuit C_{sub} that produces a non-q-Horn formula

of C_{sub}. Thus, $\phi_{sub} = \phi_1 \cdot \phi_2$, where

$$\begin{aligned}\phi_1 &= (x_5 + \overline{x_3})(x_5 + \overline{x_4})(\overline{x_5} + x_3 + x_4) \\ \phi_2 &= (\overline{x_3} + x_1)(\overline{x_3} + x_2)(x_3 + \overline{x_1} + \overline{x_2})\end{aligned}$$

Proposition 4.1 *The formula $\phi_{sub} = \phi_1 \cdot \phi_2$ is not q-Horn.*

Proof: To prove that ϕ_{sub} is not *q-Horn*, we prove by contradiction that $Z(\phi_{sub}) > 1$. Suppose that $Z(\phi_{sub}) \leq 1$. Consider $LP(\phi_1)$

$$\begin{aligned}1 + x_5 - x_3 &\leq Z \\ 1 + x_5 - x_4 &\leq Z \\ 1 - x_5 + x_3 + x_4 &\leq Z \\ 0 \leq x_3 \leq 1, \quad 0 \leq x_4 \leq 1, \quad 0 \leq x_5 &\leq 1\end{aligned}$$

Substituting $Z = 1$ in the above and solving gives

$$x_3 = x_4 = x_5 = 0 \tag{4.1}$$

Now, consider $LP(\phi_2)$

$$\begin{aligned}1 + x_1 - x_3 &\leq Z \\ 1 + x_2 - x_3 &\leq Z \\ 2 + x_3 - x_1 - x_2 &\leq Z \\ 0 \leq x_1 \leq 1, \quad 0 \leq x_2 \leq 1, \quad 0 \leq x_3 &\leq 1\end{aligned}$$

Substituting $Z = 1$ and the solution of $LP(\phi_1)$ from Equation 4.1 in $LP(\phi_2)$ gives no solution.

$$\Rightarrow Z(\phi_1 \cdot \phi_2) = Z(\phi_{sub}) > 1$$

■

From Proposition 4.1, Theorem 4.1 and Definition 4.6 it follows that any formula of which ϕ_{sub} is a sub-formula cannot be q-Horn. Thus, any circuit containing circuit C_{sub} as a sub-circuit cannot have a q-Horn CIRCUIT-SAT formula. Further, any ATPG-SAT formula derived from this circuit will not be q-Horn. Clearly circuit C_{sub} is a fairly simple circuit pattern that could be expected to occur in a large number of real circuits. Thus it appears that the practical tractability of ATPG-SAT cannot be explained by the intrinsic tractability of the SAT formulas. The answer lies in relating the solution process of SAT to properties of the circuits from which they were derived. We investigate this further in Section 4.3.

4.2.2 k-bounded Circuits

Fujiwara [Fuj88] introduced the notion of *k-bounded* circuits and showed that ATPG can be efficiently performed on this class[1]. This class was shown to contain some circuits of practical interest such as ripple-carry adders, decoders, and one- and two-dimensional cellular arrays.

Briefly, a circuit is k-bounded if its nodes can be partitioned into disjoint blocks such that each block has at most k inputs, and the blocks form a directed acyclic graph with **no reconvergent paths**. Simply put this means that all the reconvergence of the circuit is of a *local* nature, *i.e.* confined within k-input blocks. Practical circuits with deep reconvergent paths are abundant. Hence, k-boundedness seems too restrictive for general VLSI circuits.

4.2.3 Average-Time Analysis

Another approach of assessing the practical complexity of ATPG-SAT is to perform an average running time analysis on the population of ATPG-SAT instances. A number of average-time analyses already exist for different models of SAT formulas and algorithms

[1]This algorithm, described in [Fuj88], is exponential in k, but for constant k the algorithm is polynomial in the circuit size.

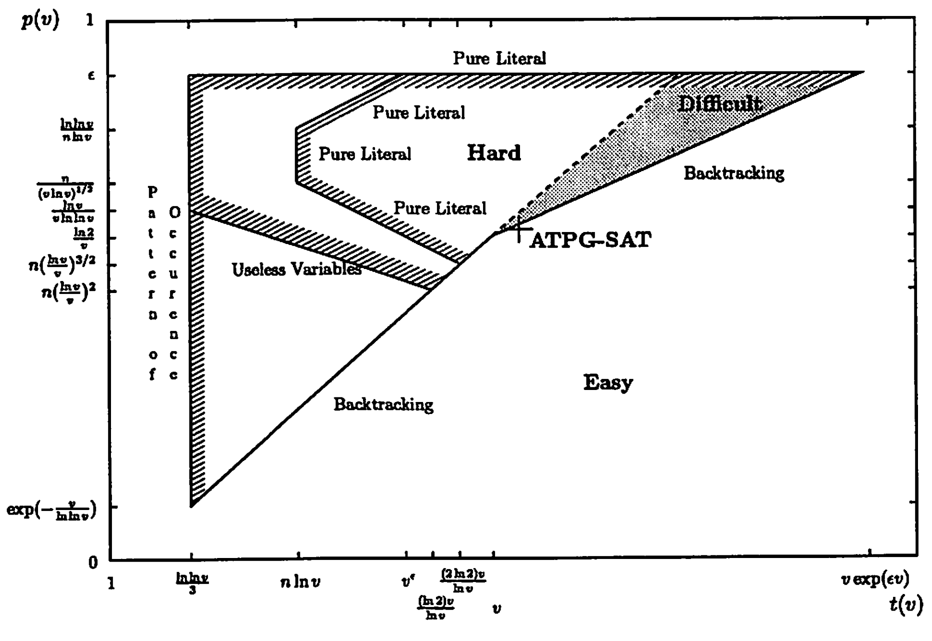


Figure 4.4: Average Time Analysis (from [PB87])

[GPFW97, PB87]; for our analysis we use the one in [PB87], since this model best matches our *CIRCUIT-SAT* problem domain.

Consider SAT instances generated by the following model. Using the notation from [PB87], let v be the number of variables in a SAT instance. Let $p(v)$ be the probability that a given literal appears in a given CNF clause, and let $t(v)$ be the number of CNF clauses which appear in the SAT instance. A given pair of functions $p(v)$ and $t(v)$ characterize a family of SAT instances.

Figure 4.4 is taken from [PB87] and depicts the space of SAT problems as a function of $p(v)$ and $t(v)$. The lines delimit areas of SAT problems which have a known polynomial *average running time*[2] algorithm and are labeled with the name of the associated algorithm. The areas labeled "Hard" and "Difficult" characterize the problems for which there is no known polynomial *average running time* algorithm.

In Figure 4.4 notice the point corresponding to $p(v) = \frac{7}{6v}$ and $t(v) = 1.963v$ which is marked with a $+$; consider the region corresponding to $p(v) = \frac{7}{6v}$, $t(v) \geq 1.963v$. This region lies in the space of random formulas that are solvable in polynomial average runtime by backtracking based algorithms. In the following we refer to this region as Γ and show

[2] *i.e.* running time averaged over all members of a class of instances.

that CIRCUIT-SAT formulas, under suitable *weak* assumptions on the underlying circuit, correspond to problems which lie within the region Γ.

Theorem 4.2 *Let C be a circuit consisting solely of 2-input AND gates, allowing inversions on the inputs and outputs of the gates. Suppose C has at most $0.528n$ primary inputs, where n is the number of gates in C. Then CIRCUIT-SAT(C) lies in the region Γ.*

Proof: The following characterization can be found in [PB87]. Let $p(v)$ and $t(v)$ be defined as above. Let $b = \lim_{v\to\infty} vp(v)$, let d be the solution of $\ln(1+d) + d\ln(1+\frac{1}{d}) = 2b$, and let ϵ be any small positive number. If $b \geq \ln 2$ and $t(v) \geq (\ln 2+\epsilon)v\frac{d}{b}$, then simple backtracking will give a polynomial average running time for the set of SAT problems corresponding to the given $p(v)$ and $t(v)$.

Note that for the problem *CIRCUIT-SAT*(C), $v = n + k$, where n is the number of gates in the circuit and k is the number of primary inputs. Given our assumption on the AND-gate decomposition of the circuit, a single gate will give rise to exactly three CNF clauses in the SAT formula. For instance, a gate for $x = y \cdot z$ corresponds to the clauses $(y+\overline{x})(z+\overline{x})(\overline{y}+\overline{z}+x)$. Two of these clauses have exactly two literals, while the remaining clause has three literals. Thus the average clause length in the overall SAT formula is $\frac{7}{3}$ literals. Since there are $2v$ possible literals, any given literal has probability $\frac{7}{6v}$ of appearing in any given clause, so $p(v) = \frac{7}{6v}$.

Thus for the circuit C, $b = \lim_{v\to\infty} vp(v) = \frac{7}{6} > \ln 2$, and solving for d gives $d = 3.305$. In the limit as $\epsilon \to 0$, we require $t(v) \geq \ln(2)v\frac{d}{b} = 1.963v$ for a polynomial average running time. But since each gate gives rise to three clauses, $t(v) = 3n$, so for a polynomial average running time we require $3n \geq 1.963(n + k)$, or $k \leq 0.528n$. Thus *CIRCUIT-SAT*(C) lies in Γ. ■

It is reasonable to assume that a large fraction of practical circuits satisfy the two conditions of Theorem 4.2. The first condition requires that the circuit consists of only 2-input AND gates, allowing inversions. Note that any circuit can be decomposed in this manner. In fact, as noted in Section 4.1, this is a decomposition technique which is commonly used for SAT-based ATPG algorithms, since it simplifies the construction of the SAT formula. Second, we assume that the primary inputs to the circuit contribute only a *small* fraction (≤ 0.528) of the variables of the SAT problem. This is also reasonable. Moreover, the process of mapping the circuit to simple 2-input AND gates will replace single complex gates with several AND gates, while keeping the number of primary inputs the same. Thus, for circuits found in practice, the number of gates is expected to be significantly greater than the number of primary inputs.

ATPG instances can be formulated as a CIRCUIT-SAT problem. Moreover, it is easily seen that if the circuit from which the ATPG instance was derived satisfies the conditions of Theorem 4.2, so will the derived ATPG-SAT instance. Thus, based on the above arguments we can claim that a large fraction of real-life ATPG-SAT instances can be expected to lie in region Γ.

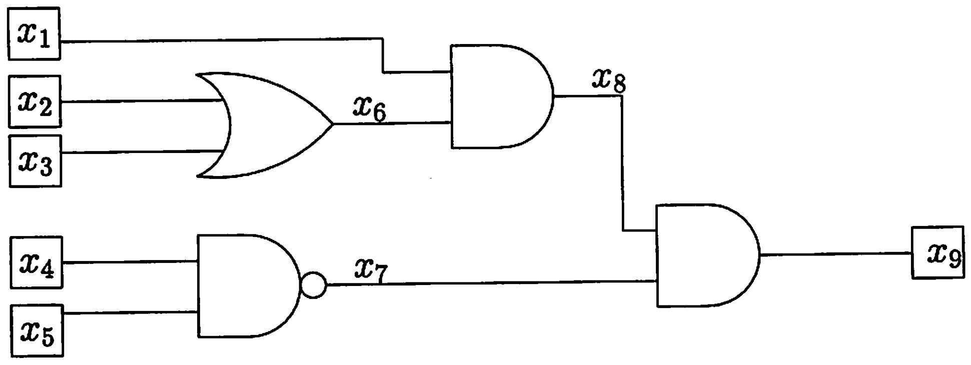
Despite this characterization, we cannot decisively conclude that practical ATPG-SAT instances can be solved in polynomial average time; while region Γ contains a large fraction of practical ATPG-SAT instances, the same $p(v)$ and $t(v)$ characteristics encompass many other SAT instances, including instances outside of CIRCUIT-SAT, and we may only conclude a polynomial average running time over the entire set of instances spanned by Γ.

Thus, this form of average-time analysis, which is representative of the state of the art in the realm of average time complexity analysis of SAT formulas, is not strong enough to prove anything conclusive about the average time complexity of real-life ATPG-SAT instances.

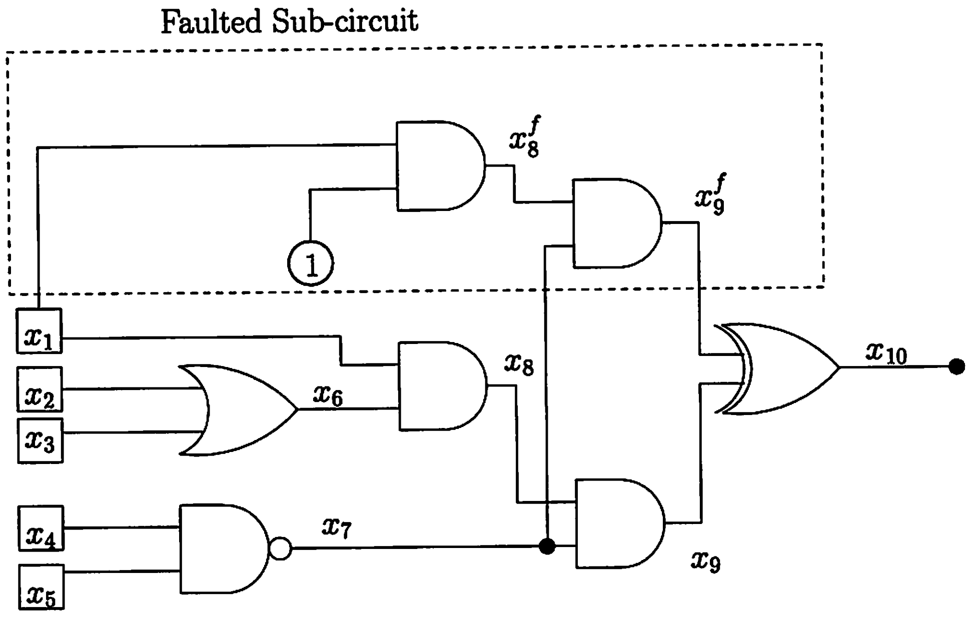
4.3 Analysis of *ATPG-SAT*

A number of approaches for solving SAT have been proposed in the literature (see [GPFW97] for a comprehensove survey). Among these, backtracking techniques based on the DPLL algorithm are the most popular. Hence, for our analysis of ATPG-SAT we chose to model the SAT algorithm by a "caching based" variant of *simple backtracking* [GPFW97]. This algorithm is described in Section 4.3.1. Briefly, the algorithm is derived from the DPLL algorithm by excluding the pure literal and unit literal rules, including the caching feature (described later) and restricting the order of all variable assignments to conform to a fixed static order.

We introduce the notion of *cut-width* of a circuit and characterize the worst case complexity of solving ATPG-SAT instances in terms of the cut-width of circuits from which the instances were derived. To illustrate the salient results, we will use the circuit shown in Figure 4.5(a) as our working example. As per the discussion in Section 2.7.2 the CIRCUIT-



(a) CIRCUIT-SAT example



(b) ATPG circuit example

Figure 4.5: Example Circuits

SAT instance corresponding to this circuit is:

$$(\overline{x_2}+x_6)(\overline{x_3}+x_6)(x_2+x_3+\overline{x_6})(x_4+x_7)(x_5+x_7)(\overline{x_4}+\overline{x_5}+\overline{x_7})$$
$$(x_1+\overline{x_8})(x_6+\overline{x_8})(\overline{x_1}+\overline{x_6}+x_8)(x_8+\overline{x_9})(x_7+\overline{x_9})(\overline{x_8}+\overline{x_7}+x_9)(x_9) \qquad (4.2)$$

The ATPG problem we consider is a *stuck-at-1* fault on the net x_6. The ATPG-SAT instance generated by this fault corresponds to the circuit shown in Figure 4.5(b).

4.3.1 Caching-Based Backtracking for CIRCUIT-SAT

Our *caching based version* of simple backtracking is a simplified way of modeling the notion of learning from previous conflicts. This notion is implemented as *conflict-clause recording* [MSS99] in almost all current successful SAT solvers. The essential idea of caching based backtracking is to perform simple backtracking with a fixed variable order, except that whenever the algorithm backtracks from an unsatisfiable sub-formula, the sub-formula is cached. Correspondingly, before a sub-formula is taken up for a satisfiability check, it is looked up in the cache. If found, it can be diagnosed immediately as being unsatisfiable and the algorithm can backtrack from it without trying any further variable assignments. The pseudo-code for the algorithm appears below. In Algorithm 4.1, ϕ is the CNF formula for the satisfiability check, h is a function that orders the variables of ϕ, and $\mathcal{L}$ is a hash table for storing the set of *unsatisfiable* sub-formulas of ϕ encountered during the backtracking search.

Figure 4.6 shows an example run of this algorithm on Formula 4.2. The variable ordering $\mathcal{O}_1 = (x_2 < x_3 < x_6 < x_1 < x_8 < x_4 < x_5 < x_7 < x_9)$ is used for the backtracking search. Note there are several places where the caching strategy works to prune the search. For example, consider the partial assignment $x_2 = 0, x_3 = 0, x_6 = 0, x_1 = 0, x_8 = 0$; this leaves the sub-formula $(x_4+x_7)(x_5+x_7)(\overline{x_4}+\overline{x_5}+\overline{x_7})(x_7+\overline{x_9})(\overline{x_9})(x_9)$. This same sub-formula is obtained under the assignment $x_2 = 0, x_3 = 0, x_6 = 0, x_1 = 1, x_8 = 0$ and so we can prune this branch of the search without further computation.

The running time of Algorithm 4.1 on a given formula ϕ, is denoted by $\mathcal{T}(\phi)$ and can be analyzed as follows. A *sub-formula* of ϕ is obtained by setting a subset of the variables of ϕ to certain values. Define a *consistent sub-formula (CSF)* of ϕ as a *sub-formula* having no *empty clauses*[3] (*i.e.* a clause where all the literals have been set to *false* under the partial assignment).

[3]A formula with empty clauses is trivially unsatisfiable.

Algorithm 4.1 Satisfiability through Caching-Based Backtracking

```
procedure Sat($\mathcal{L}, h, \phi$)
$\mathcal{L} \leftarrow \emptyset$
if Cache_Sat($x_{first}, 0, \phi, h$) = "UNSAT" and Cache_Sat($x_{first}, 1, \phi, h$) = "UNSAT" then
  return "UNSAT"
else
  return "SAT"
end if


procedure Cache_Sat($x_{current}, \mathcal{B}, \phi_{sub}, h$)
{$x_{current}$ : Variable currently chosen for assignment, $\mathcal{B}$ :Value assigned to $x_{current}$}
$\phi_{sub} \leftarrow$ Assign($\phi_{sub}, x_{current}, \mathcal{B}$)
if Null_Clause($\phi_{sub}$) then
  return "UNSAT"
else {$\phi_{sub}$ has no NULL clauses}
  if Table_Lookup($\mathcal{L}, \phi_{sub}$) then
    return "UNSAT"
  end if
  $x_{next} \leftarrow$ Next_Var($x_{current}, h$)
  if Cache_SAT($x_{next}, 0, \phi_{sub}, h$) = "SAT" then
    return "SAT"
  end if
  if Cache_SAT($x_{next}, 1, \phi_{sub}, h$) = "SAT" then
    return "SAT"
  end if
  {Both Subtrees UNSAT}
  Insert_Table($\mathcal{L}, \phi_{sub}$)
  return "UNSAT"
end if
```

Ordering $\mathcal{O}_1 : x_2 < x_3 < x_6 < x_1 < x_8 < x_4 < x_5 < x_7 < x_9$

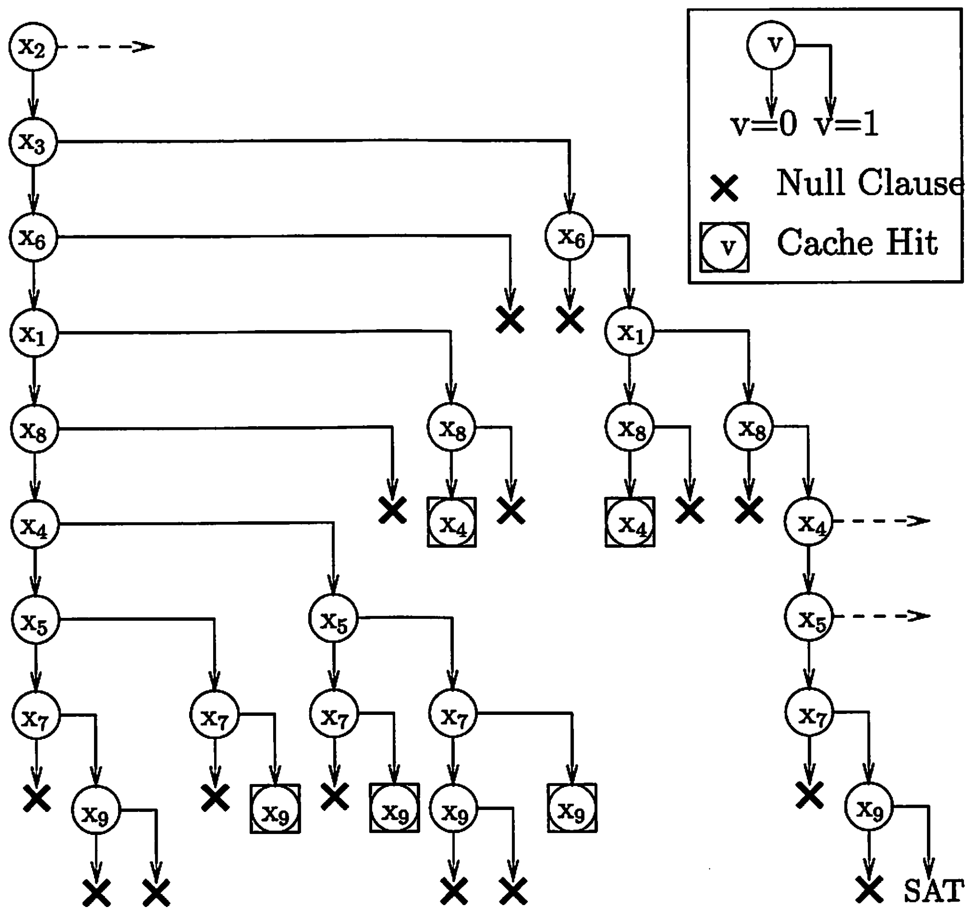


Figure 4.6: Caching-based backtracking for Formula 4.2

We assume that the sub-formulas are cached as sets of clauses. Thus, from our point of view two sub-formulas are identical if and only if they have the same set of clauses. Sub-formulas with a different set of clauses may still be functionally equivalent; however, we do not recognize this equivalence in this treatment. $\mathcal{T}(\phi)$ is upper bounded by the product of the size of the backtracking tree (*i.e.* the search tree explored by the SAT algorithm) and the worst case time for a single cache access (insertion, lookup or deletion). For the purpose of this analysis we assume that the caching is perfect; cache lookups and insertions can be done in constant time[4]. Thus, $\mathcal{T}(\phi)$ is upper-bounded by the size of the backtracking tree, which in turn is bounded by the number of *distinct consistent sub-formulas (DCSFs)* of ϕ that can be generated under a particular static ordering of the formula variables. Thus, under the ordering h,

$$\mathcal{T}(\phi) = O(\mathcal{F}(\mathcal{P}_h(V))) \tag{4.3}$$

where $\mathcal{F}(\mathcal{P}_h(V))$ is the number of DCSFs of ϕ under the ordering h, V is the set of variables of ϕ and $\mathcal{P}_h(V)$ denotes the set of those subsets of V which are valid prefixes of the ordering h. If the formula ϕ corresponds to a CIRCUIT-SAT instance, generated from a circuit C, we can further characterize $\mathcal{T}(\phi)$ in terms of a topological property of C. This characterization is developed in the following section.

4.3.2 Cut-width and Sub-formula Count

Consider a *CIRCUIT-SAT* formula $\phi(C)$ corresponding to circuit C. For the initial part of the analysis assume that C has a single output. The results are extended to multi-output circuits, in Section 4.3.3. The network C can be seen as an undirected hypergraph with the signals as the hyper-edges, and the gates, inputs and outputs as the nodes. For the purpose of this exposition a Boolean network and its underlying hypergraph are not distinguished. *Cut-width* of a hypergraph is defined as follows.

Definition 4.7 (Cut-width) *Given a hypergraph $G(V, E)$ and a one-to-one function h, ordering the vertices of G. $h : V \rightarrow \{1, 2, \ldots, |V|\}$. The cut-width of G, under the ordering h, is denoted as $W(G, h)$ and is given by the expression*

$$W(G,h) = \max_{i \in \{1,2,\ldots,|V|\}} |\{e \in E : \exists u, v \in V \\ \textit{such that } \{u,v\} \subseteq e \textit{ and } h(u) \leq i < h(v)\}|$$

[4]An imperfect cache can add to the overall complexity by a linear factor at worst.

(**Note:** Each hyperedge e of G is denoted by the set of vertices spanned by that hyperedge.) The *minimum cut-width* of G over all possible orderings h is denoted by $W_{min}(G)$. Henceforth, cut-width of a circuit without reference to a particular variable ordering will refer to the minimum cut-width $W_{min}(G)$.

Figure 4.7 illustrates the notion of cut-width on the example circuit from Figure 4.5(a), using two different variable orderings, $\mathcal{O}_1$ and $\mathcal{O}_2$. Ordering $\mathcal{O}_1$, which was used for the backtracking tree example of Figure 4.6, also happens to be a minimum cut-width (W_{min}) ordering for this circuit.

The number of nodes at a certain level in the backtracking tree for $\phi(C)$ can be bounded in terms of the size of an appropriate *cut* of the circuit C. A disjoint partition $(\delta_{V_C}, \overline{\delta_{V_C}})$ of the variables V_C defines a unique *cut* in C. An assignment of truth values to the variables δ_{V_C} in the formula $\phi(C)$ yields a sub-formula $\phi_{sub}(C)$ of $\phi(C)$.

Lemma 4.3 *Given a Boolean network C, its corresponding CIRCUIT-SAT formula $\phi(C)$ and a* cut *$(\delta_{V_C}, \overline{\delta_{V_C}})$ of V_C, the number of DCSFs that can be obtained by the set of all possible truth assignments to the variables δ_{V_C} is denoted by $\mathcal{F}(\delta_{V_C})$ and can be bounded as:*

$$\mathcal{F}(\delta_{V_C}) \leq 2^{2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})|} \tag{4.4}$$

where $|(\delta_{V_C}, \overline{\delta_{V_C}})|$ denotes the size of the cut, i.e. the number of distinct nets crossing the cut.

Proof: Consider the set of $2^{|\delta_{V_C}|}$ possible different Boolean assignments to the variables δ_{V_C}. Only a fraction of these produce consistent sub-formulas. Consider only these assignments. They partition the clauses of $\phi(C)$ into three *disjoint* categories.

- Clauses all of whose variables are part of δ_{V_C}. Every CSF of $\phi(C)$ has these clauses satisfied.
- Clauses all of whose variables are part of $\overline{\delta_{V_C}}$. These clauses are unaffected by any assignment to the variables δ_{V_C} and thus appear unaltered in *any* consistent sub-formula.
- Clauses part of whose variables are in $\overline{\delta_{V_C}}$. We call these clauses *injured clauses.*

From the above categorization it is clear that different consistent sub-formulas of the set $\mathcal{F}(\delta_{V_C})$ differ only in the configuration of the *injured clauses.* Furthermore, under any

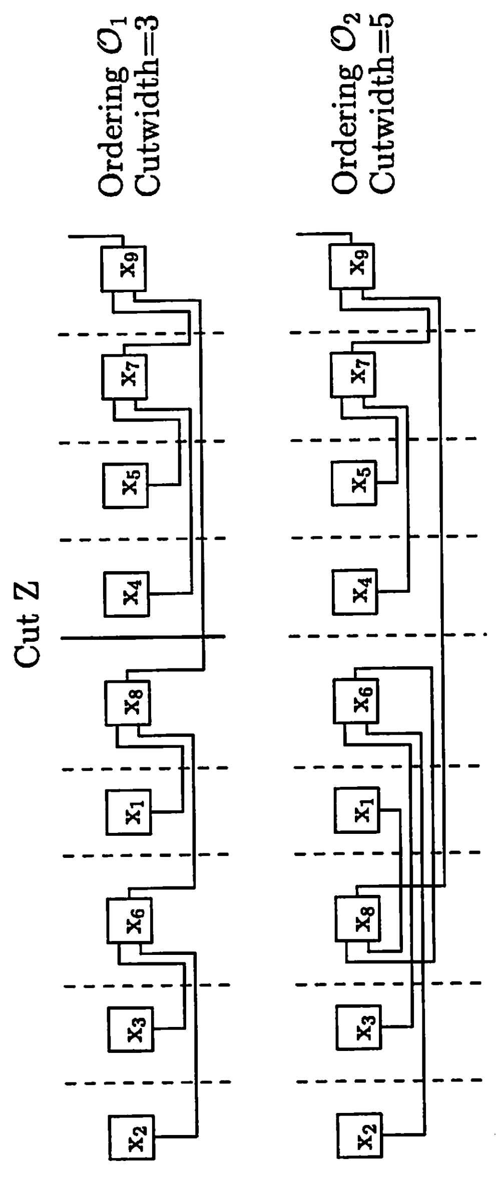


Figure 4.7: Example cut-widths for the circuit of Figure 4.5(a)

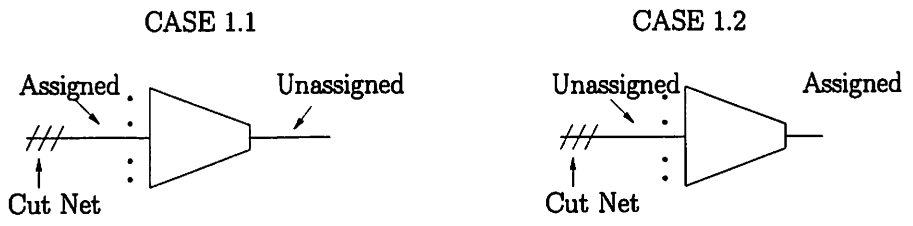


Figure 4.8: Case 1 for generating injured clauses

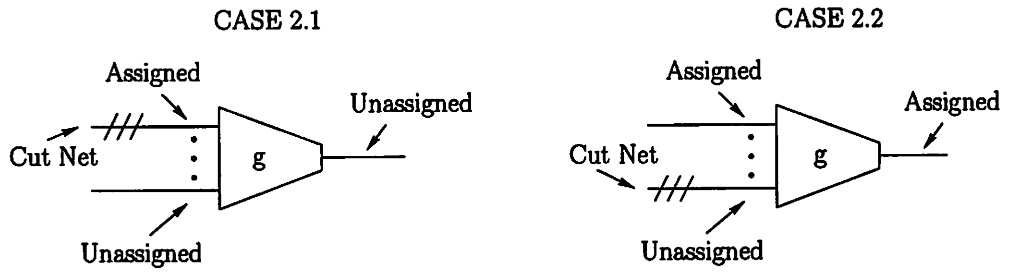


Figure 4.9: Case 2 for generating injured clauses

assignment to variables δ_{V_C} an injured clause can have only two different configurations. Consider a typical injured clause $\omega = (l_1 + l_2 + \ldots + l_i + l_{i+1} + \ldots + l_k)$ such that the variables corresponding to literals $l_1 + l_2 + \ldots + l_i$ are part of δ_{V_C} and the remaining variables (corresponding to literals $l_{i+1} + \ldots + l_k$) are part of $\overline{\delta_{V_C}}$. Under any assignment to the variables δ_{V_C}, ω takes one of the two configurations, $(l_{i+1} + \ldots + l_k)$ or 1 (*i.e.* it has been satisfied). Thus we can bound

$$\mathcal{F}(\delta_{V_C}) \leq 2^{(\#\text{ injured clauses})} \tag{4.5}$$

The number of injured clauses can be upper bounded as follows. Every injured clause must contain ***at least*** one assigned variable and at least one unassigned variable. Moreover, a pair of variables occur in a common clause only under one of the following two cases:

- **Case 1:** They form an input-output pair for a gate (see Figure 4.8). For this pair to produce an injured clause either the input variable is assigned and the output unassigned or vice-versa. In both these cases, the input net falls in the cut $(\delta_{V_C}, \overline{\delta_{V_C}})$.
- **Case 2:** They form a pair of "sibling" inputs for a common gate g (see Figure 4.9). As before, they can be responsible for an injured clause if and only if one of them

is assigned and the other unassigned. Additionally, the output of g can be either assigned or unassigned. In either case, from the clause construction of Figure 2.2, it is clear that every injured clause that these "siblings" participate in already contains a pair of variables that have been counted in Case 1 (namely the output of g and the input that differs from the output in assignment status). Thus, this case is subsumed by Case 1.

From the above case analysis it is evident that *every* injured clause can be associated with a cut-net and also that Case 1 can account for all injured clauses. Since the network is fanout-bounded by k_{fo}, each cut net can fan out to at most k_{fo} gates and therefore produce at most k_{fo} instances of Case 1. Moreover, since the network is composed of simple gates only, a given pair of variables can occur in at most two common clauses (see Figure 2.2). Thus each cut net can account for at most $2k_{fo}$ injured clauses. Hence,

$$\text{Number of injured clauses} \leq 2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})| \tag{4.6}$$

Applying this result to Equation 4.5 the bound on $\mathcal{F}(\delta_{V_C})$ follows.

■

The usefulness of this result stems from the fact that the formula set size is exponential not in the size of the variable set but in the size of the cut, which could be potentially much smaller. For example consider the cut $(\delta_V, \overline{\delta_V})$ on the circuit of Figure 4.5(a), with $\delta_V = \{x_2, x_3, x_6, x_1, x_8\}$; this corresponds to the level in the backtracking tree corresponding to the *Cut Z* label in Figure 4.7. Lemma 4.3 indicates that there can be at most 2^2 distinct consistent sub-formulas generated by all possible value assignments to the δ_V variables, whereas a naive bound would be 2^5 (there are 2^5 distinct assignments to the variables δ_V).

Based on the above we derive the following bound for the running time of Algorithm 4.1.

Theorem 4.4 *Given a Boolean network C and ordering h on V_C, Algorithm 4.1 can solve the CIRCUIT-SAT instance $\phi(C)$ in time $O(n \cdot (2^{2k_{fo}W(C,h)}))$, where $n = |V_C|$.*

Proof:

To prove the result, we derive a bound on $\mathcal{F}(\mathcal{P}_h(V_C))$ and then apply Equation 4.3. Recall that $P_h(V_C) = \{\delta_{V_C} | \delta_{V_C} \subseteq V_C$, and δ_{V_C} is a prefix of the ordering $h\}$. Therefore $|\mathcal{P}_h(V_C)| = |V_C| = n$.

$$\begin{aligned}
\mathcal{F}(\mathcal{P}_h(V_C)) &\leq \sum_{\delta_{V_C} \in \mathcal{P}_h(V_C)} \mathcal{F}(\delta_{V_C}) \\
&\leq n \cdot \max_{\delta_{V_C} \in \mathcal{P}_h(V_C)} \mathcal{F}(\delta_{V_C}) \\
&\leq n \cdot \max_{\delta_{V_C} \in \mathcal{P}_h(V_C)} 2^{2k_{fo}|(\delta_{V_C}, \overline{\delta_{V_C}})|} \\
&\qquad \text{(from Lemma 4.3)} \\
&= n \cdot 2^{2k_{fo}W(C,h)} \text{ (from Definition 4.7)}
\end{aligned}$$

■

From the above result it is evident that if a circuit has a cut-width which is *logarithmic* in the size of the circuit, CIRCUIT-SAT can be performed on it in polynomial time. We discuss further implications of this result in Section 4.4.

As explained in Section 4.1, under the SAT formulation of the ATPG problem, testing for a certain fault ψ on a circuit C amounts to performing CIRCUIT-SAT on a certain circuit, namely C_ψ^{ATPG}. The following result shows that, for any fault ψ in circuit C, the cut-width of C is linearly related to the cut-width of C_ψ^{ATPG}. This means that we can reason about the asymptotic behavior of Algorithm 4.1 on ATPG-SAT instances generated from circuit C by analyzing the cut-width properties of circuit C (or sub-circuits thereof) rather than having to deal with the circuit C_ψ^{ATPG}.

Lemma 4.5 *Given a Boolean network C, for any ordering h of the variables V_C and any fault ψ on C , $\exists$ an ordering h_ψ of the variables of C_ψ^{ATPG} such that*

$$W(C_\psi^{ATPG}, h_\psi) \leq 2 \cdot W(C,h) + 2 \tag{4.7}$$

Proof: The circuit C_ψ^{ATPG} is composed of the two sub-circuits C_ψ^{fanout} and $C^{sub,\psi}$ and a single 2-input XOR gate y (see Figure 4.2). Note that both C_ψ^{fanout} and $C^{sub,\psi}$ are sub-circuits of C. One may see that given any variable ordering h for V_C, this implies a corresponding ordering h_{sub} for any sub-circuit C_{sub} of C such that $W(C_{sub}, h_{sub}) \leq W(C,h)$.

Given ordering h for V_C, h_ψ can be constructed as follows. Extract the implied orderings for sub-circuits C_ψ^{fanout} and $C^{sub,\psi}$ from h. Now merge these two together by putting each variable x^f of the faulted sub-circuit C_ψ^{fanout} just after its corresponding "unfaulted variable" x (derived from $C^{sub,\psi}$). Now construct h_ψ by adding y to the beginning

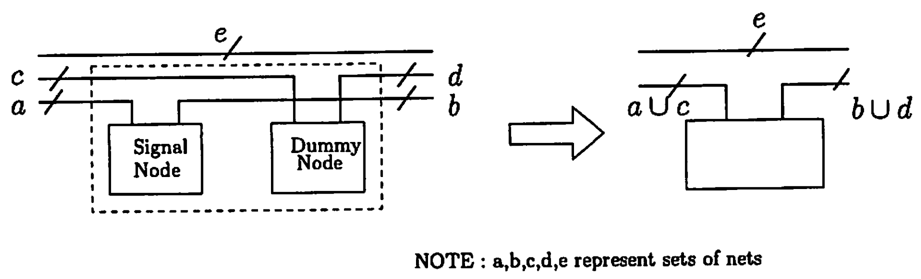


Figure 4.10: Proof for ATPG circuit width vs. original circuit width

of this merged ordering. To derive the width of C_{ψ}^{ATPG} under this ordering, consider the following.

The gate y is a two-input gate and can contribute at most 2 to $W(C_{\psi}^{ATPG}, h_{\psi})$ (this accounts for the additive 2 in the expression). Now, for the moment assume that the primary inputs of C_{ψ}^{fanout} are not fed from fanout points in $C^{sub,\psi}$ but from separate dummy nodes (the dummy nodes are inserted after the corresponding signal nodes in the ordering h_{ψ}). In this scenario it is easy to see that the width of the resulting circuit is at most $2 \cdot W(C, h) + 2$. Merging the dummy nodes with the corresponding signal nodes does not increase the cut-width of the resulting circuit (see Figure 4.10). Hence the required result follows. ■

Figure 4.11 illustrates this result on our example ATPG circuit from Figure 4.5(b). As shown in Figure 4.7 the circuit of Figure 4.5(a) has a cut-width of 3 under ordering $\mathcal{O}_1$ (Figure 4.6). The ordering $\mathcal{O}_1'$ can be derived (see the proof of Lemma 4.5 above) from $\mathcal{O}_1$ to yield a cut-width of 4 for the ATPG circuit of Figure 4.5(b).

4.3.3 Extension to Multi-output Circuits

The discussion so far has been restricted to single-output circuits. Consider a multi-output circuit C, with p primary outputs $o_1, o_2, \ldots o_p$. For the purpose of a CIRCUIT-SAT test, C can be seen as a set of p single-output circuits $\{C_1, C_2, \ldots C_p\}$, one each for the *transitive fanin cone* of each primary output. CIRCUIT-SAT on C can be performed by performing CIRCUIT-SAT on each of the single-output circuits $C_1, C_2, \ldots C_p$, one at a time. Then, *CIRCUIT-SAT*(C) = *CIRCUIT-SAT*(C_1) + ... + *CIRCUIT-SAT*(C_p).

In this scenario, the results of Sections 4.3.1 and 4.3.2 can be applied to multi-output circuits as follows. Given a multi-output circuit $C = \{C_1, C_2, \ldots C_p\}$ and a set

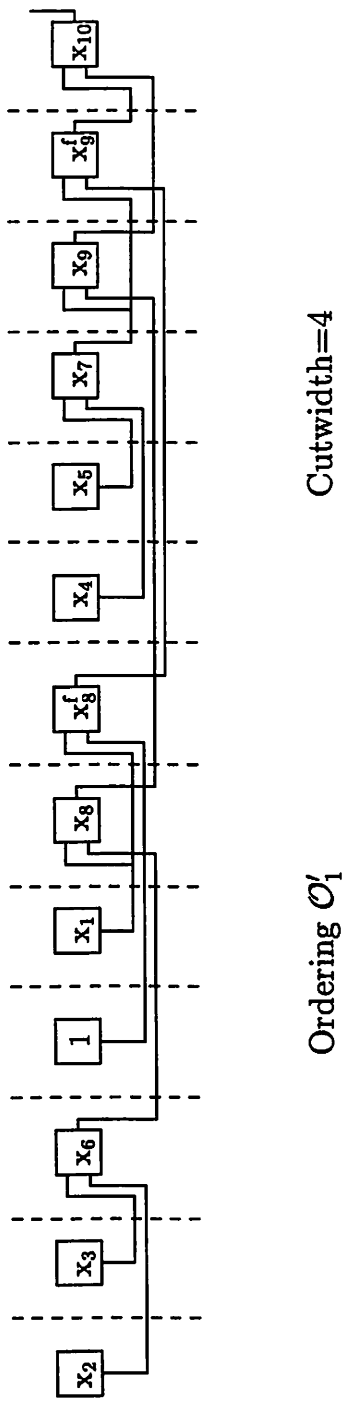


Figure 4.11: Example cut-width, ATPG circuit of Figure 4.5(b)

$H = \{h_1, h_2, \ldots h_p\}$ of node orderings for the single-output circuits $C_1, C_2, \ldots C_p$, the notion of cut-width as given by Definition 4.7 can be extended as:

$$W(C,H) = \max_{i \in \{1,2,\ldots p\}} W(C_i, h_i) \tag{4.8}$$

The minimum cut-width $W_{min}(C)$ generalizes on similar lines, except now the minimum is over all possible *sets of orderings* H. Hence the running time of *CIRCUIT-SAT*(C), (based on Algorithm 4.1) can be bounded as:

$$\mathcal{T}(\phi(C)) = O(p \cdot n_{max} \cdot 2^{2k_{fo}W(C,H)}) \quad \text{where } n_{max} = \max_{i \in \{1,2,\ldots p\}} |V_{C_i}| \tag{4.9}$$

Similarly, Lemma 4.5 can be restated as:

Lemma 4.6 *Given a multi-output Boolean network C, for any set of orderings $H = \{h_1, h_2, \ldots h_p\}$ of the variables $V_{C_1}, V_{C_2}, \ldots V_{C_p}$ and any fault ψ on C, $\exists$ an ordering H_ψ of the variables of C_ψ^{ATPG} such that*

$$W(C_\psi^{ATPG}, H_\psi) \leq 2 \cdot W(C,H) + 2 \tag{4.10}$$

4.4 Cut-width Properties of Circuits

4.4.1 Log-bounded-width Circuits

In the following we define a class of circuits known as *log-bounded-width circuits* and show that by employing Algorithm 4.1 ATPG can be efficiently performed on these circuits. We also prove that k-bounded circuits (see Section 4.2.2) lie within the class of log-bounded-width circuits.

Definition 4.8 (Log-bounded width circuit) *A given multi-output circuit C is log-bounded-width if for each single stuck-at fault ψ on C, there exists a set of orderings H of the variables $V_{C^{sub,\psi}}$, such that*

$$W(C^{sub,\psi}, H) = O(log(|C^{sub,\psi}|)) \tag{4.11}$$

Theorem 4.7 *Given a log-bounded-width circuit C and any single stuck-at fault ψ on C, test generation for ψ can be accomplished in time polynomial in the size of the circuit C.*

Proof: Applying Lemma 4.6 and Equation 4.9 to the definition of log-bounded-width above (Definition 4.8), we can use Algorithm 4.1 to solve the instance *ATPG-SAT*(C, ψ) in time polynomial in $|C^{sub,\psi}|$. Since $|C^{sub,\psi}| \leq |C|$, then the running time is polynomial in $|C|$ as well. ∎

Lemma 4.8 *Given a k-ary tree T over n vertices, there exists an ordering h, of the variables V_T such that $W(T,h) \leq (k-1)\log(n)$.*

Proof: Consider a k-ary tree T over n vertices with root r. For a vertex ordering, take the variables by using depth-first search starting from the root; at each node visit the children in increasing order of the size of the sub-trees rooted at each child. Under this ordering T has a max-cut of at most $(k-1)\log(n)$ edges. This can be proved by induction on n. For the base case $n = 1$, the cut is zero.

For larger n, assume $W(T,h) \leq (k-1)\log(m)$ for trees of size $m < n$. The induction proof has two cases. Let s_i, $1 \leq i \leq k$ be the subtrees rooted at the immediate children of r, and let c_i be the size of the max-cut for s_i. For the first case, let all $|s_i| \leq n/2$. Then the max-cut under the given DFS ordering is at most $(k-1)+c$, where $c = \max_i c_i$. By the induction hypothesis, $c \leq (k-1)\log(n/2)$, so the max-cut of T is at most $(k-1)\log(n)$.

For the second case, for some t, $|s_t| > n/2$ and there can be at most one of these. Then this subtree is visited last by the DFS ordering, and so the max-cut of T by this ordering is at most $\max((k-1)+c, c_t)$, where $c = \max_{i \neq t} c_i$. Since $|s_i| < n/2, i \neq t$, the induction hypothesis gives $(k-1)+c \leq (k-1)\log(n)$, and since $|s_t| < n$, $c_t \leq (k-1)\log(n)$ as well.

Thus the max-cut of T is at most $(k-1)\log(n)$. ∎

Theorem 4.9 *Any k-bounded circuit, for a given constant k is log-bounded-width.*

Proof: First consider the graph G consisting of the blocks of a k-bounded circuit; by the non-reconvergence property of k-bounded circuits, the cone for each output of G is a k-ary tree. For each output tree, use the ordering scheme proposed in the proof of Lemma 4.8 to order the blocks of G. Now, within each block order the vertices of the block arbitrarily. Each block can thus increase the max-cut by a factor of at most 2^k. Hence, given Lemma 4.8 for k-ary trees we can conclude an upper bound of $2^k \cdot (k-1)\log(n)$ for the max-cut of a k-bounded circuit. ∎

As shown above, tree circuits are of log-bounded-width. Intuitively, reconvergence tends to increase circuit cut-width. But, as long as the circuits are sufficiently "tree-like" the log-bounded-width property could be expected to apply. The *locality of reconvergence* required by k-boundedness is just one instance of this, (which log-bounded-width has been shown to capture). In principle log-bounded-width simply requires a *minimality of reconvergence* and is therefore a more general property than k-boundedness. Note that Theorem 4.7 applies to *all* faults in the circuit, including the redundant faults, which need to be proven untestable.

4.4.2 Practical VLSI circuits

It is clear that cut-width is intrinsically linked to the topology of the circuit. Thus, when a class of circuits can be described in terms of suitable topological characteristics, it is possible to derive the cut-width properties of that class, and therefore reason about the asymptotic complexity of ATPG-SAT, as was done for *log-bounded-width circuits* and *k-bounded circuits* above. However, practical designs are usually not specified in such a manner. Moreover, extracting common topological characteristics from a set of arbitrary circuit designs is non-trivial and beyond the scope of this research. Thus, we have instead performed an empirical study of cut-width for a set of circuits. The study is organized in two parts. First we study circuits in the **MCNC91** and **ISCAS85** multi-level combinational benchmark suites, estimate their cut-widths and compare the cut-widths to the size of the circuit.

[HGRC98] presents a system which extracts topological properties from a given circuit and generates arbitrarily large circuits which have similar characteristics. In the second part of our study, we use this scheme to generate a "family" of circuits from a given circuit and then examine the cut-width properties of this family.

4.4.2.1 Experimental Setup

The key element of our experimental setup is a mechanism to measure the *cut-width* of a single-output circuit C. This can then be used to derive the cut-width of a multi-output circuit. From the definition, the minimum cut-width is simply the value of the *max-cut* obtained under a *min-cut linear arrangement* [GJ79] of C. Since the min-cut linear arrangement (MLA) problem is known to be NP-complete, it would not make sense to challenge the exponential complexity of ATPG if the procedure for deriving the ordering for the ATPG (in other words, the cut-width) involved the solution of another NP-Complete problem, namely MLA. Therefore, we use a well-known polynomial time algorithm [Hoc97] to approximate the MLA, and hence upper-bound the cut-width for a given circuit. Since our entire empirical analysis and conclusions are based on the approximate cut-width, the actual complexity of exact MLA has no bearing on our results or conclusions. It is noteworthy that practical ATPG tools often use some kind of topological ordering for the branch and bound. In many cases this actually coincides with an optimal cut-width ordering, for example in the case of trees. Therefore, even though the ATPG tool may not be working with the cut-width metric in mind while deriving the ordering it may serendipitously generate a cut-width optimal (or close to optimal) ordering.

Our approximation algorithm for MLA generates a placement based on recursive mincut bipartitioning, until the partitions are sufficiently small and then performs an exact MLA for each of these partitions. We used the **HMETIS** package [KAKS99] from the University of Minnesota to perform the bipartitioning.

For each benchmark circuit the complete set of all stuck-at-0 and stuck-at-1 faults was first pruned by using fault-collapsing methods (*viz.* fault-dominance and fault equivalence). Random vector generation was not used to further reduce the fault list in order to keep the data set interestingly large and rich for the following data-analysis. Note that this does not bias our results in any way since our analysis is a worst-case analysis, and any worst-case efficiency result derived on a set of faults would certainly hold on any pruned subset of it[5].

For each fault of the collapsed set, one data point was generated as follows. For a given fault ψ in circuit C, the data-point measures the approximate cut-width of the circuit $C^{sub,\psi}$ versus the size of this circuit. The size of the circuit $C^{sub,\psi}$ is in direct correspondence to the size of the SAT instance $ATPG\text{-}SAT(C,\psi)$ (in terms of the number of variables) and the cut-width of this circuit is representative of the complexity of solving this instance (as per Equation 4.9 and Lemma 4.6).

[5]The fault collapsing based on dominance and equivalence just removes multiple, *identical* superimposed points from the plots which do not add any real value to the results.

4.4.2.2 Study of Existing Benchmark Suites

In studying the cut-width properties of the **MCNC91** and **ISCAS85** benchmark circuits, it became clear that the individual circuits have different structural properties. Some have nodes with fanin of a dozen or more inputs, while others are composed solely of two-input AND gates and inverters. Similarly, some of the benchmark circuits have nodes which implement complex functions, while others use only simple AND/OR gates.

These differences probably would not exist in actual implementations of circuits; fanin and node complexity is necessarily limited due to speed and size requirements on the gates. Moreover, in performing ATPG it is often desirable to map circuits to simple AND and OR gates (with inverters), since the corresponding SAT formulas become easier to derive. Thus, in order to bring more uniformity to the circuits and to more closely emulate the actual ATPG process, we mapped the benchmark circuits to three or fewer input AND/OR gate networks (allowing inversions) using the `tech_decomp` procedure from the **SIS** [S+92] package.

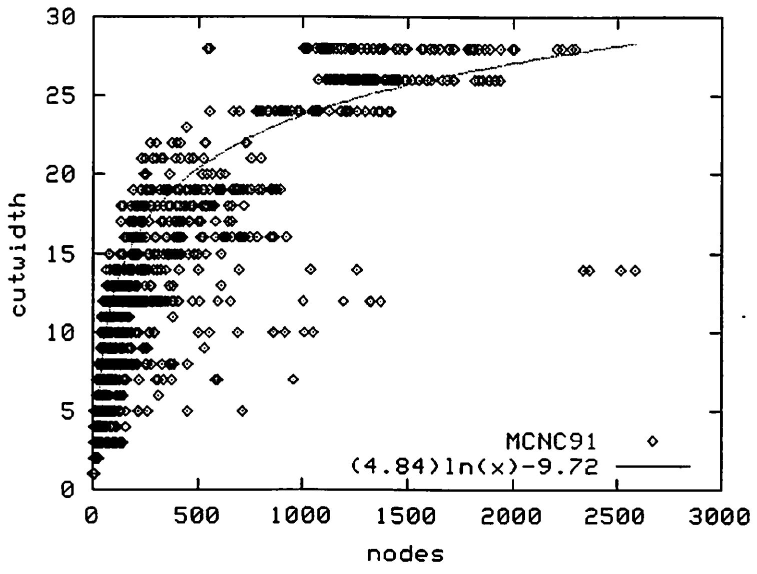
Figure 4.12(a) shows the results for the circuits identified as "logic" circuits from the **MCNC91** benchmark suite. We excluded circuit `t481`, which we considered degenerate, having over 3800 nodes after gate mapping yet with only a single output. Figure 4.12(b) corresponds to the **ISCAS85** combinational benchmark circuits. We omitted the circuits `C3540` and `C6288` in this analysis, due to limitations in our min-cut linear arrangement tool.[6] We expect `C6288` to have a large cut-width.

In any event, our method ran successfully for all the remaining benchmarks (48 from **MCNC91** and 9 from **ISCAS85**).

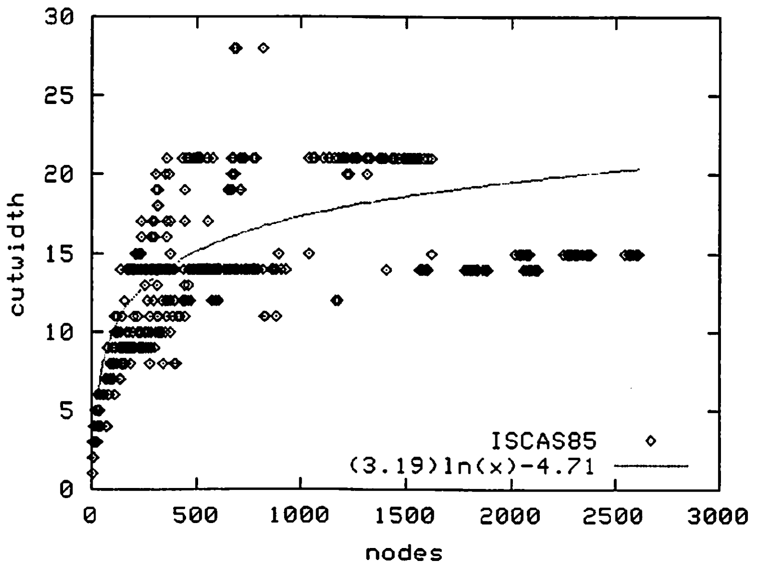
4.4.2.3 Study of Generated Circuits

Using the existing benchmark suites limits the size of the circuits which we analyze. Ideally, we would like to have a large range of circuit sizes so that we can examine the growth of the cut-width with larger circuits. To this end, we use *synthetic benchmark generation* techniques to construct example circuits over a wide range of sizes. These techniques take existing circuits, extract statistical properties deemed critical to producing "realistic" circuits, and generate random circuits with these same characteristics. [HGRC98]

[6]We used HMETIS in a mode which fixed some vertices to specific partitions. These circuits generated too many fixed vertices for HMETIS to handle.



(a) MCNC91 logic benchmarks (47786 datapoints)



(b) ISCAS85 benchmarks (15096 datapoints)

Figure 4.12: Cut-width results for benchmark circuits

and [GB99] propose methods for synthetic benchmark generation. We chose to apply the programs `circ` and gen, described in [HGRC98], to generate our benchmark examples. Note that these programs do not generate any useful circuits in that the function of each node is undefined. However, this is of no concern, since we are only interested in the structure of the circuit, in particular the cut-width corresponding to each possible fault point.

Our goal is to generate circuits with structures similar to the original MCNC benchmark circuits but with varying sizes. To this end, we use `circ` to find the characteristics for a selection of the benchmark circuits, and then scale these parameters before using gen. In particular, we change only the number of nodes in the circuit, the number of primary inputs, the number of primary outputs, and the number of edges (nets) in the circuit. We do not change the depth of logic, since this parameter is bounded for practical circuits to meet delay constraints. We also do not change the distribution of delays, fanouts or edge lengths in the circuit; [HGRC98] identifies these parameters as important in characterizing the structure of a circuit, and we wish to obtain circuits structurally similar to the original benchmarks.

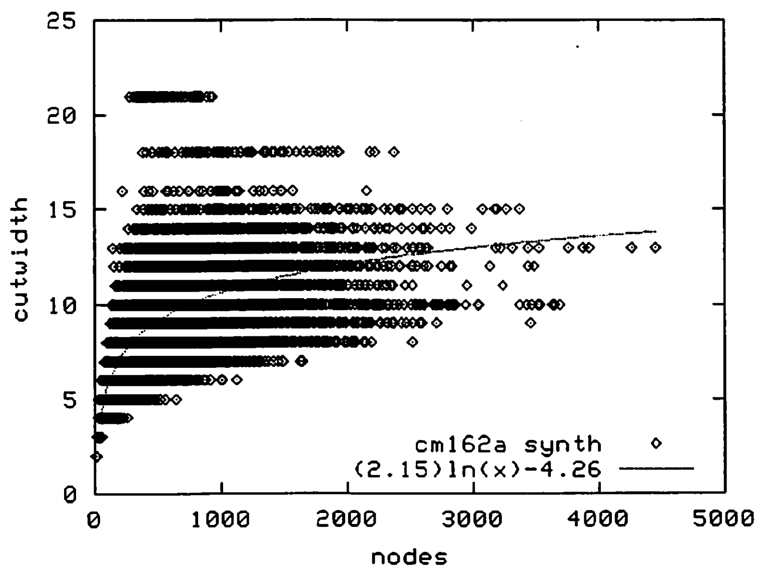
For each benchmark circuit used here, we used `circ` and gen to generate a "family" of circuits ranging from $1,000$ nodes to $6,000$ nodes. For each generated circuit, we take each possible fault ψ, find the induced sub-circuit $C^{sub,\psi}$, and calculate the size of this sub-circuit and estimate its minimum cut-width; this is exactly the same procedure as used with the original benchmark circuits.

Figures 4.13(a) through 4.14(b) show the cut-width versus circuit size for four different families of circuits generated as described above.

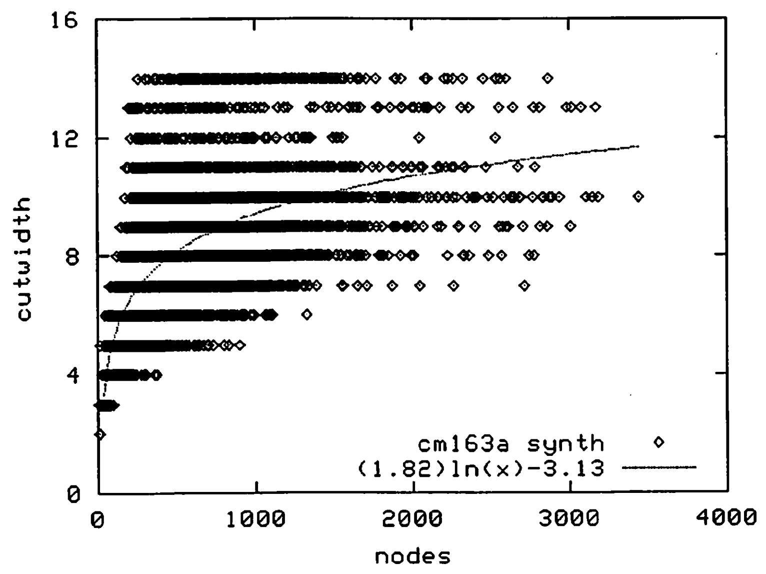
4.4.2.4 Experimental Results

The cut-width plots for the MCNC91 and ISCAS85 suites, and the four families of cloned circuits (Figures 4.12(a) through 4.14(b)) reveal several interesting properties of real-life circuits. First, the cut-width values saturate at values of around 10-20 for all six sets of benchmarks. This is even true for data-points with circuit-sizes of *thousands* of nodes. Thus, Theorem 4.4 shows that the complexity of solving these instances would be of the order of 2^{20} as against 2^{1000} which is what a naive worst case analysis would predict.

Secondly, it is evident from all six plots that the cut-width is a slowly growing function of the circuit size. To ascertain the precise functional nature of the growth we

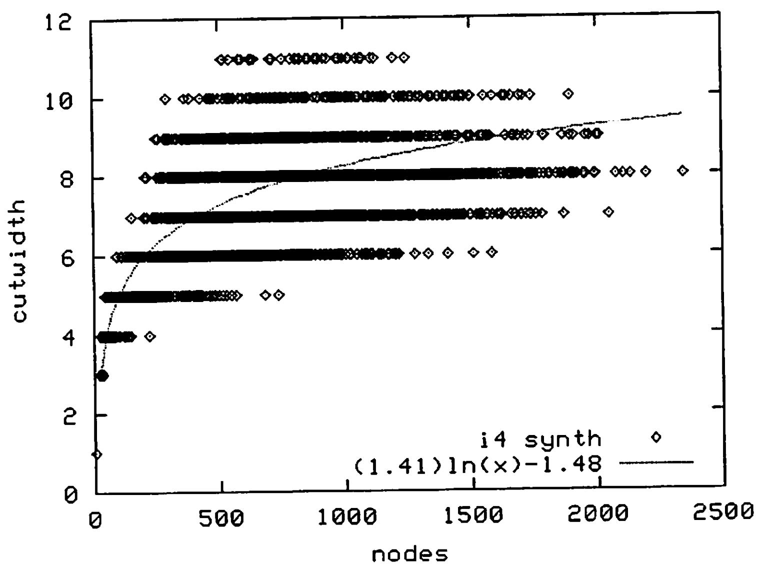


(a) cm162a family (37192 datapoints)

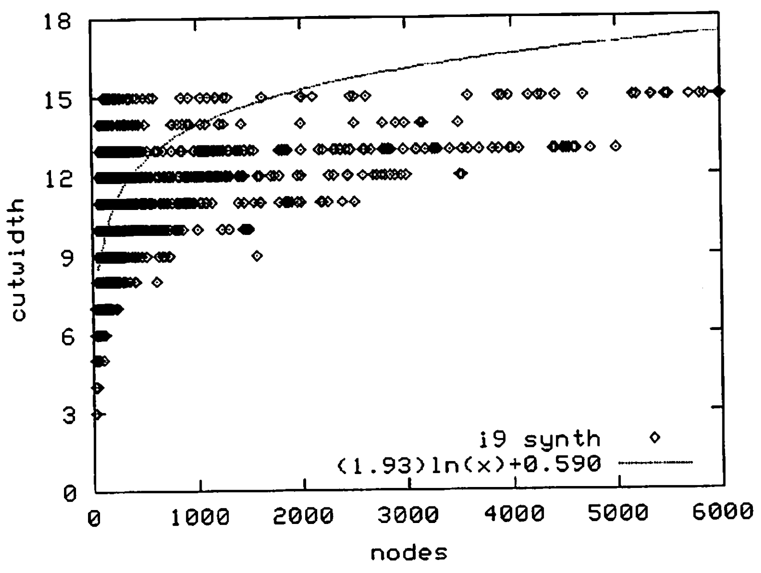


(b) cm163a family (37776 datapoints)

Figure 4.13: Cut-width results for generated circuits



(a) i4 family (38297 datapoints)



(b) i9 family (37215 datapoints)

Figure 4.14: Cut-width results for generated circuits

Circuit Set	$ax+b$	$a\sqrt{x}+b$	ax^b (b)	$a\log x+b$
MCNC91	9.060	4.982	5.013(0.425)	**4.085**
ISCAS85	2.930	2.467	2.015(0.339)	**2.015**
i9 family	1.447	1.273	1.202(0.216)	**0.984**
i4 family	0.573	0.476	0.445(0.244)	**0.429**
cm162a family	2.363	1.969	1.861(0.299)	**1.727**
cm163a family	1.204	0.936	0.865(0.298)	**0.812**

Table 4.1: Sum of squared errors for various functional fits on each data set (Normalized by a factor of 10^5)

performed curve-fitting on the six data sets. While there is no *provably correct* procedure of determining the precise functional nature of a set of data, the accepted practice [Mey75, Ric95] consists of the following two-step procedure:

1. A candidate distribution (say $f(x)$) is chosen, based on a combination of visual inspection of plotted data and theoretical prediction.

2. The exact parameters of $f(x)$ for the given data are determined by performing a fit based on the *Least Squares Error* [Mey75]. Alternatives to the least squares metric are known but the least squares error method is by far the most popular one.

In case there are multiple candidates for functions suiting the data the best functional fit can be found by comparing the *squared error* value of the best fit for each of the candidate functions.

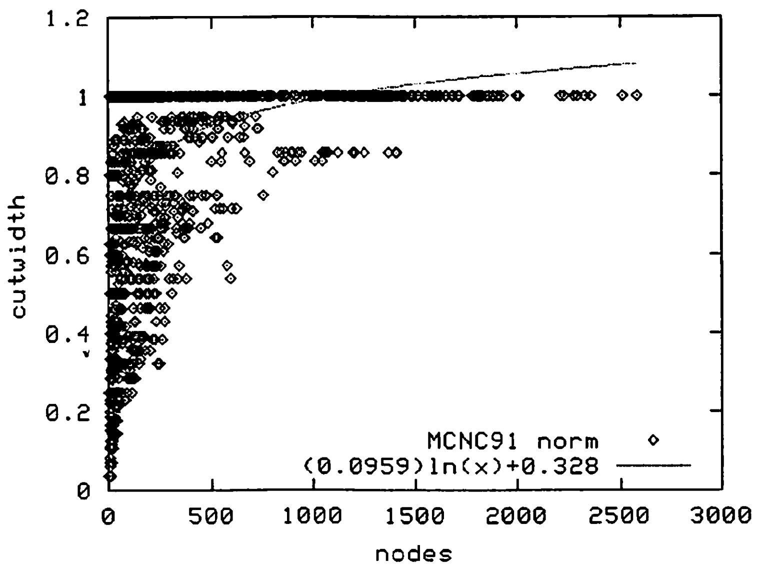
In conformity with this procedure, we used a least-squares method [Mey75] to fit four different functions to each of the six data sets: linear ($f(x) = ax + b$), square-root ($f(x) = a\sqrt{x} + b$), power ($f(x) = ax^b$) and logarithmic ($f(x) = a\log(x) + b$) curves, where $f(x)$ denotes the cut-width and x is the number of nodes. A super-linear function can be ruled out since, by definition, cut-width can be no larger than the size of the circuit. The squared-error for the four functional fits is listed in Table 4.1. Of the four curves, the log curve gives the smallest square error for all six benchmark sets; the best-fit log curves are shown in the figures. These plots suggest that the cut-width is indeed a logarithmic function of circuit size for these circuits, and so we can expect these benchmarks to be easily testable. This agrees with the empirical results from **TEGUS** (Figure 4.1).

The functional fit depicted by the above plots is fairly convincing, especially considering the following *noise sources* which might be present. First, we are attempting to fit discrete data to a continuous function. Hence, there would be some noise on account of *truncation errors*.

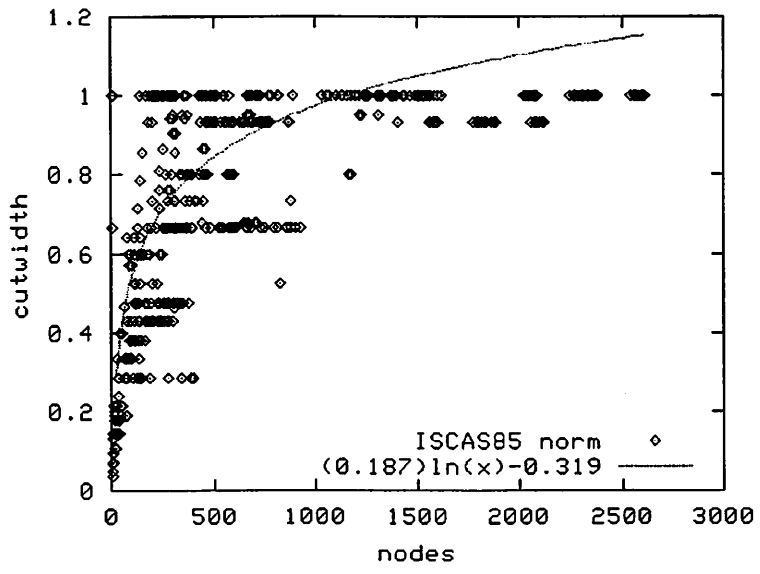
Second, since the datapoints come from different circuits it is conceivable that they in fact lie on a family of related curves rather than a single curve. Thus, there would be a *normalization error* when we try to fit the data to a single curve. An effort has been made to partially mitigate this effect by working with sets of circuits with similar topology. However, this effect still surfaces in some cases where the plots appear to give the semblance of a family of curves rather than a single curve. Another way to correct for this error could be use a normalization scheme on the data, before curve-fitting. We have experimented with a few simple normalization schemes. The results of one such scheme on the ISCAS and MCNC cut-width data sets of Figures 4.12(a) and 4.12(b) is shown in Figures 4.15(a) and 4.15(b). Here, the cut-widths from each circuit are normalized with respect to the maximum saturation cut-width value obtained from that circuit. This scheme does appear to partly correct the normalization error for these plots, but it is not as effective in the case of the synthetic circuits. In some cases the effect of these errors can be strong enough to completely mask out any kind of pattern in the cut-width data. Such is the case with the *ISCAS89 sequential circuits* which we tried to analyze. We have therefore omitted those results from this study.

In a nutshell, since curve fitting procedures have traditionally not been applied to the application at hand, *i.e.* for the asymptotic complexity analysis of a combinatorial algorithm they are not tuned to deal with these problems. In principle, these techniques could be adapted and tuned to further sharpen the curve fit and the conclusions we have tried to derive above. However, such an effort would require some research into curve fitting techniques per se, which is beyond the scope of the analysis presented here.

In any case, the growth of the cut-width is definitely sub-linear with the size of the circuit. The value of the exponent, b of the power curve fit (recorded in parentheses in Table 4.1), which consistently assumes a value between $0.2 - 0.35$ (*i.e.* less than 1) for all six sets of data, further buttresses this fact. Thus, while the logarithmic nature of the cut-width growth provably gives a polynomial runtime for ATPG (Theorem 4.7) from an asymptotic complexity standpoint, in practice the *slow* growth of cut-width is sufficient to ensure that the complexity of ATPG on typical circuits grows sub-exponentially with the



(a) MCNC91 logic benchmarks



(b) ISCAS85 benchmarks

Figure 4.15: Normalized cut-width results

size of the problem.

4.5 BDDs and CIRCUIT-SAT

The concept of *circuit-width* has been used by researchers [Ber91, McM92] to obtain upper bounds on the size of BDDs representing the circuit function. At first glance our treatment of circuit cut-width would seem to bear a striking similarity to these results. However, our results have no direct relationship to the BDD bounds. We discuss this aspect in some detail below and conclude that neither result subsumes the other, each useful in its own domain.

Binary decision diagrams (BDDs) and CNF Boolean formulas are both representations of Boolean functions. Solving CIRCUIT-SAT on a Boolean circuit C could be done by building a BDD for the circuit and doing a "0" check on the BDD. Alternatively, one can construct a CNF Boolean formula $\phi(C)$ and solve satisfiability on the formula using a backtracking algorithm. In essence, a BDD and a backtracking tree represent the same entity, *i.e.* the Boolean space of the function.

Berman [Ber91] gave a bound on the BDD size, for any *topological ordering* of the circuit elements. This result was extended by McMillan [McM92] for arbitrary orderings. McMillan's result can be summarized as follows. Given a single-output circuit C, with n inputs, if the elements of C can be linearly ordered such that over all cross-sections of the linear arrangement, w_f (forward width) bounds the number of wires running in the forward direction and w_r (reverse width) bounds the number of wires in the reverse direction, then the size of the BDD representing the output of C can be upper bounded by $n2^{w_f 2^{w_r}}$. This result differs from the result presented in this paper on two counts.

- Our definition of circuit cut-width is independent of the direction of signal-flow (our characterization of width is on an undirected hypergraph) and thus substantially different from w_f and w_r in an operational sense.
- The above result is exponential in the *forward width* and double-exponential in the *reverse width*, while our result has only a single exponential. We exploit this property in defining the class of log-bounded-width circuits.

The explanation for these discrepancies lies in the following differences between BDDs and CIRCUIT-SAT formulas. BDDs represent the intrinsic nature of a Boolean function, independent of the specific hardware implementation, while CIRCUIT-SAT formulas (as per the construction of Section 4.1) are in one to one correspondence with the circuit topology. The result of [McM92] bounds the BDD size by bounding the number of possible multi-output functions that a certain sub-circuit of the original circuit could compute. Our proof technique however, treats the SAT formula as a string encoding the circuit topology and tries to bound the number of distinct sub-strings that can be generated from a partial truth assignment to the CIRCUIT-SAT variables. Therefore, the two results, although similar in spirit, characterize different entities altogether.

4.6 Conclusions

We have presented a worst case complexity analysis for a SAT based formulation of the combinational ATPG problem which accounts for salient characteristics of problem instances encountered in real life. Incidentally, this work is also one of the first attempts at reconciling the theoretical, worst case complexity of combinational ATPG with the relative ease with which practical instances of it are solved. For the purpose of analysis we have employed the SAT based ATPG formulation proposed by Larrabee [Lar92], with a caching based variant of simple backtracking (see Section 4.3) used to model the SAT solver.

Under this model of the algorithm the complexity of ATPG on a given circuit has been characterized in terms of a topological property of the circuit, namely the *undirected circuit cut-width*. Theoretical arguments and experimental results confirm that this property can be used to predict polynomial runtimes of ATPG, for a wide range of practical VLSI circuits.

Specifically, this analysis has been used to define a class of circuits called log-bounded-width circuits which we have shown to be efficiently testable. Additionally, this class of circuits has been shown to subsume the class of k-bounded circuits. Our experiments on a wide range of benchmark and generated circuits show that they exhibit the log-bounded-width property. On an intuitive level the log-bounded-width property essentially captures the "treeness" of the circuit. As long as a circuit has limited reconvergence (not necessarily local reconvergence), the log-bounded-width property can be expected to apply.

Practical ATPG engines [SBSV96, Lar92] employ a host of other search pruning techniques to reduce complexity, such as random test pattern generation *etc.* [ABF95], and in some cases these techniques do bring about substantial reduction in complexity. The benchmark C6288 from the ISCAS85 suite is a notable example which has a large cut-width but is efficiently testable through random test pattern generation. However, such cases are more the exception than the rule and a cut-width based argument is much more generally applicable and can single handedly account for the tractability of ATPG on a wide range of real-life instances. Nevertheless, the presented analysis can be improved by including more algorithmic features of ATPG and SAT tools in the algorithm model used for the analysis.

It is noteworthy that although the present analysis is aimed at the combinational ATPG problem, the notion of cut-width and its use in analyzing the complexity of a CIRCUIT-SAT based problem formulation is not endemic to the current problem. The same analysis framework can potentially be modified to apply to other EDA problems.

Chapter 5

SAT-Based Combinational Equivalence Checking

Combinational equivalence checking (CEC) is one of the most of widely used formal techniques in the verification of digital circuits. While, theoretically the problem is co-NP Hard, practical instances of the problem are more tractable. Current design methodology ensures that the two combinational circuits being checked for equivalence have a fair degree of structural and functional similarity [BT89]. In recent years several approaches to CEC have been proposed which exploit the above property. While these techniques have significantly advanced the state of the art in CEC, the inherent complexity of the problem and the growing size and complexity of digital systems continues to motivate further research.

Most of the successful programs for CEC use a combination of various engines, with Binary Decision Diagrams (BDDs) [Bry86] as the main workhorse. Although a few of the proposed approaches use Boolean Satisfiability (SAT) [MSG99] or SAT-like engines (*viz.* ATPG methods [Bra93], recursive learning [Kun93]) as the principal engine, these methods have not become popular. Consequently, the use of SAT in current CEC is largely ancillary to BDDs; *e.g.* it is used to eliminate false negatives or to choose candidate pairs for deducing intermediate relationships [BS98].

The work presented in this chapter makes a case for the use of SAT methods in CEC. There are several reasons for pursuing this line of research. First, there have been significant advances in SAT algorithms [Stå, MSS99, MMZ$^+$01]. Second, while it has been claimed that BDDs are relatively more efficient for CEC, neither has a quantitative comparison been published nor the reasons for the purported inefficiency of SAT algorithms analyzed in detail. Third, as discussed in more detail in Section 5.3, SAT algorithms have several inherent features which BDDs lack, that can potentially make them a more flexible and robust core technology for this application. This raises the following questions:

- Is the perceived inefficiency of SAT algorithms in CEC a necessary consequence of the use of SAT algorithms per se, or is it an artifact of the particular SAT algorithm and the way it was used in the CEC framework?
- Is it possible to bridge the efficiency gap between SAT-based and BDD-based CEC tools by using more sophisticated SAT algorithms currently available and/or by fine-tuning the implementation of the tool[1]?

This research addresses these questions. The main contributions of the work presented here can be summarized as follows:

- We present a detailed analysis of the features of SAT algorithms and BDDs in the context of CEC to argue that SAT based algorithms can be a more flexible and robust core technology in this application.
- We present a simple CEC framework drawing from a number of previously proposed CEC methodologies [Bra93, BS98, vE97] as well as our own insights into applying SAT for CEC. SAT algorithms form the core engine of this approach.
- We make a direct quantitative comparison between a preliminary implementation of the proposed CEC framework and a state-of-the-art BDD-based, mixed-engine tool for CEC [BS98], and assess the performance gap between BDD-based and SAT-based checkers.
- We offer insights into several avenues for improving the performance of the above SAT based tool. In our opinion, with these enhancements, the proposed SAT-based checker has the potential to outperform state-of-the-art BDD-based CEC tools.

The experiments reported in Section 5.5 show that our checker outperforms state-of-the-art SAT-based CEC methods by over two orders of magnitude. Moreover, even the

[1]BDD-based tools draw upon over a decade of research in variable ordering and efficient implementation, as well as highly tuned implementations of CEC packages, while precious little has been done in these respects for SAT in CEC.

current prototype implementation is only moderately slower (a factor of 2-3) and sometimes faster than state-of-the-art BDD-based mixed-engine checkers. This work is presented as a proof of concept to show how SAT-based techniques can effectively remedy the inherent problems associated with BDD-based methods. We advocate that once suitably tuned and applied, SAT-based techniques can more *actively* complement and even replace BDDs in CEC while significantly advancing the state-of-the-art in this area.

The rest of the chapter is organized as follows. Section 5.1 reviews the CEC problem and the modern view of a general framework to solve it in a practical setting. Section 5.2 briefly discusses previous efforts in the areas of BDD-based and SAT-based CEC. In Section 5.3 we provide arguments and illustrations to show how SAT-based methods can potentially be a more flexible and robust tool for Boolean reasoning in CEC. Section 5.4 describes our proposed SAT-based CEC framework. Experimental results comparing our method with several existing SAT-based CEC tools as well as a state-of-the-art BDD-based mixed engine CEC tool are presented in Section 5.5. Section 5.6 concludes the chapter with a discussion of several avenues for improving the performance of the proposed CEC framework.

5.1 The Combinational Equivalence Checking Problem

Let C_1 and C_2 be two combinational logic circuits with the same set of primary inputs, denoted by $I = i_1, i_2, \ldots i_n$, each having a single primary output (assumed for ease of exposition), denoted by o_1 and o_2 respectively. The *combinational equivalence checking (CEC)* problem or *combinational verification* problem over C_1 and C_2 is to determine if both circuits implement the same logic function *i.e.* if for each of the 2^n Boolean value assignments to inputs I, o_1 and o_2 evaluate to the same logic value.

Although in general this problem is co-NP Hard, in practice the circuits C_1 and C_2 exhibit a fair degree of structural and functional similarity [BT89]. In recent years, this property has been exploited to develop powerful engines for combinational equivalence checking [BT89, Bra93, BS98, KK97, Kun93, MSG99, Mat96, PK00]. Most of these approaches operate under the following general framework.

The similarity between the two circuits is exploited to deduce specific succinct relationships (equivalences, implications, replacability relationships) between internal nodes (called *cutpoints* [KK97]) of the two circuits being checked for equivalence. Using these relationships the overall equivalence check is performed as a set of smaller equivalence checks. Briefly, a *cutpoint* is an internal node of one circuit that is proven to be related to one or more internal nodes of the other circuit through a specific succinct relationship (usually *equivalence* or *equivalence modulo inversion*). The algorithm proceeds by sweeping the two circuits (or the *miter* [Bra93]) from inputs to outputs, deducing new cutpoints from previously deduced cutpoints, until the primary outputs are proved equivalent or a miscomparing pattern is found. The algorithm maintains a *cut* or *frontier* of cutpoints deduced thus far. This is used as a basis for deducing further cutpoints. Negatives (either false or true) encountered during this process, as a result of functional constraints between internal circuit nodes, are resolved by attempting to justify them towards the primary inputs.

Overall, this methodology comprises a *Deduction Engine* to derive internal node relationships and a *Justification Engine* which eliminates false negatives or identifies true negatives. A *negative* is a witness assignment to a set of existing cut-points $x_1, x_2, \ldots x_k$ used to *disprove* the existence of a cut-point relationship (*e.g.* equivalence) between a pair of potential new cut-points. A *false negative* is a negative which cannot be justified back to the primary inputs *i.e.* there does not exist a primary input assignment under which the signal nodes $x_1, x_2, \ldots x_k$ assume the witness assignment claimed by the negative. The *false negative problem* assumes great importance in the context of the above methodology where the algorithm lacks a global view of the circuits when attempting to deduce new cutpoints. In practice considerable resources of the algorithm are devoted to efficiently resolving potential false negatives.

The following exposition will be with respect to cut-point methods based on deducing *equivalences*. Thus the cut or cut-point frontier will be referred to as an *equivalence cut* and denoted by δ. Note that an equivalence cut carries a topological interpretation in terms of C_1 and C_2 (it partitions the inputs of C_1 and C_2 from their outputs) as well a set interpretation (it is a set of variables forming the physical cut in the circuits C_1 and C_2). In the following we use both these interpretations interchangeably. The equivalence cut is also *irredundant* in the sense that no proper subset of it forms a cut over C_1 and C_2.

5.2 Previous Approaches

As described in Section 5.1 a typical cut-point based method is composed of a *deduction engine* and a *justification engine.* In many of the proposed works on CEC [JMF95, KK97, Mat96], BDDs are used in both the deduction engine as well as the justification engine. Recently, Burch and Singhal [BS98] proposed a methodology where BDDs are the primary deduction engine as well as part of the justification engine. Randomized SAT algorithms, modified to work off the BDDs are used to supplement BDD based justification. Paruthi and Kuehlmann [PK00] proposed a tighter integration of BDDs and SAT based methods for CEC using an interleaved combination of BDDs and a SAT solver as the deduction engine. However, BDDs continue to be a major part of their deduction engine. Moreover their method of using the SAT solver in the overall flow is fairly orthogonal to our proposed approach.

There have been a few attempts to use SAT based algorithms to perform the entire equivalence check. Brand [Bra93] proposed a cutpoint based methodology based on *replacability relationships* which were derived using an ATPG tool. **HANNIBAL** [Kun93] used *recursive learning* to derive implications which were then used by an ATPG tool to perform the equivalence check. Marques-Silva *et al.* [MSeS99] proposed using a recursive learning based pre-processor to derive equivalence relationships which are subsequently used by a general purpose SAT solver to perform the verification task. While these methods offer an innovative alternative to BDD-based methods, they have not become the method of choice for CEC; generally it is believed that SAT-based methods are not as efficient as BDD-based methods. However, we believe that this is not a necessary consequence of using SAT methods vs. BDDs but rather a result of the specific SAT algorithms used and the way they have been applied in the overall methodology. This work is an attempt to validate this claim.

A number of other approaches addressing the CEC problem have appeared in the literature. However, they have been omitted from the above survey since they are not directly relevant to the focus of this research, which is the application of SAT methods to CEC. The interested reader is referred to [BS98, JNFSV97] for more detailed surveys on other CEC approaches.

5.3 SAT Vs. BDDs in CEC

As described in Section 5.2, BDDs have been successfully used as the core *equivalence deduction engine* in a number of programs for CEC [BS98, KK97, Mat96]. The reason for this success is the ability of BDDs to compactly represent the *complete Boolean space* of reasonably large functions (variable support of up to 15-20 variables or even greater).

However, SAT algorithms have an inherent advantage over BDDs in Boolean reasoning, under a given set of constraints. BDDs have no means of performing *Boolean constraint propagation (BCP)*, a feature that is integral to all branch and bound based SAT solvers. Since branching based SAT solvers explore each assignment to the variables of the formula one by one, BCP or "examining the logical consequences of each assignment", is a natural component of such algorithms. Thus, such an algorithm can actually work with (branch on) only a small portion of the given Boolean variables while still being able to examine the logical consequences of this branching on the remaining variables at a negligible additional expense (the expense of BCP). On the other hand, BDDs work by constructing a representation of the *entire Boolean space* of a specified set of output variables, in terms of a specified set of input variables. The only way to introduce additional variables is to explicitly construct BDDs of those functions as well and connect them to the existing BDDs by some logical operation, *viz.* conjunction or existential quantification. For the current application *i.e.* CEC, this single feature gives a SAT-based algorithm several operational advantages. Properly harnessed, these can translate into significant gains in the overall efficiency and robustness of the tool. Some of these are discussed below.

5.3.1 Locality and robustness of cutpoint resolution

In order to deduce an internal equivalence $x \equiv x'$ a typical BDD based deduction engine has to build BDDs of x and x' in terms of a common set of cutpoints (**Y**,**Y'**) such that $x(\mathbf{Y}) \equiv x'(\mathbf{Y'})$. To determine a suitable set (**Y**,**Y'**) such methods [BS98, Mat96, vE97] resort to a host of heuristics to *resolve* cutpoints backwards until a suitable cut is found. Such an approach is inherently unrobust since there is no good criterion to determine the "right cut" to learn an equivalence from. Thus, often such an approach uses a set (**Y**,**Y'**) much larger and farther away than needed to learn the equivalence. The key point is that the inability to learn $x \equiv x'$ from a given cut (**Y**,**Y'**) is due to the presence of certain *false negatives* on this cut. Often it is possible to resolve these through local BCP or a fairly

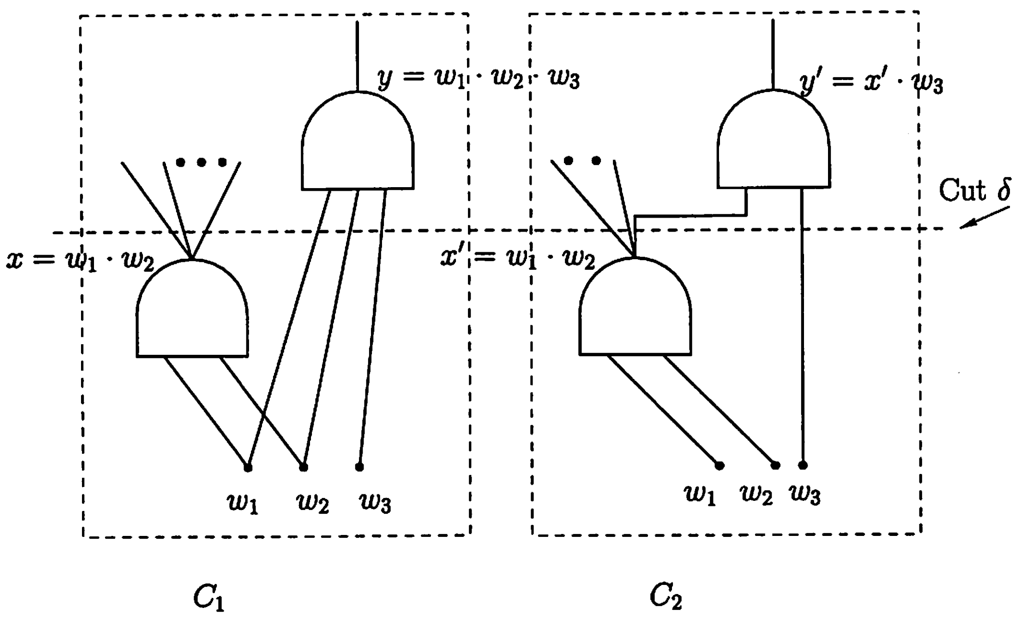


Figure 5.1: False Negatives can be resolved by local BCP

"local branch and bound search" by a SAT algorithm.

Example 5.1 *Consider the circuit of Figure 5.1 where the signals y and y' are not equivalent in terms of the cut $\delta = \{x, x'\}, w_1, w_2, w_3$ but are actually globally equivalent in terms of the signals w_1, w_2, w_3*[2]*. The miscomparing patterns (*e.g. $x = x' = 1, w_1 = 0, w_2 = 1, w_3 = 1$*) can be easily resolved by a SAT procedure through local BCP, while operating from the cut δ, but a BDD based approach operating in terms of the same cut would not be able to deduce the equivalence of $y \equiv y'$.*

5.3.2 Use of previously deduced equivalences

To the best of our knowledge all cutpoint based methods immediately merge nodes that are deduced as equivalent. With BDD based methods there is probably no benefit in doing otherwise. However, with SAT based methods it is possible to simply add the deduced equivalence as a clause or constraint to the overall formula without merging the two nodes and benefit from this.

[2]Such a situation is frequently produced by simple operations such as factoring and re-substitution in logic optimization.

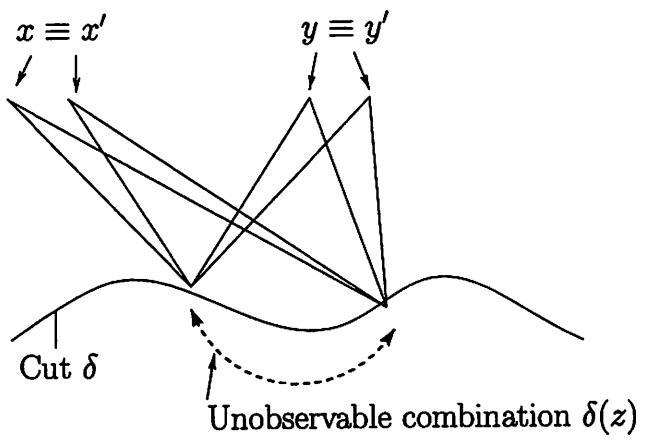


Figure 5.2: Previously deduced equivalences as shallow witnesses of false negatives

Example 5.2 *Consider the example of Figure 5.2 where the current cutpoint frontier δ has an unobservable or false negative assignment $\delta(z)$ which blocks learning equivalences $x \equiv x'$ as well as $y \equiv y'$ from the cut δ. Once we have expended some branching effort in backjustifying this false negative for learning $x \equiv x'$ we can add in $x \equiv x'$ as a* local witness *of $\delta(z)$ (and not merge x, x') so that when trying to learn $y \equiv y'$ this false negative can be justified with no branching effort (since this assignment will immediately violate $x \equiv x'$).*

The same reasoning applies to not merging equivalences behind the current equivalence cut so that they can be used as *shallow witnesses* of unobservable assignments when trying to backjustify false negatives towards the primary inputs.

5.3.3 Learning more general relationships

In almost all BDD based cutpoint methods the notion of cutpoints corresponds to equivalence relationships (or equivalence modulo inversion) in terms of the circuit primary inputs. Such relationships can be naturally obtained by BDD pointer comparisons. However, using SAT methods it is possible to work with a much more general notion of cutpoints. One such generalization [Bra93] proposed the notion of *replacability* of gates where x can be replaced with y iff on replacing x with the gate $z = x \oplus y$ there does not exist a test for the *stuck-at-0* fault at the output of z. This and a number of other variations of this notion can be realized by slightly modifying the SAT problem posed to the solver. However, for simplicity we have chosen not to exercise this degree of freedom in this work.

5.4 Proposed Methodology

Our overall framework is similar to most cutpoint based methods as described in Section 5.1. The key difference is that we use SAT procedures alone to accomplish both the deduction and the justification phases. As in [KK97] the two circuits to be checked for equivalence are decomposed into a network of two-input AND gates, allowing inversions on the edges. This decomposed network is used as the base data-structure. Currently, the deduction procedure is restricted to deducing *equivalences* ($x \equiv y$) and *equivalences modulo complementation* ($x \equiv \bar{y}$). All deduced relationships are tagged onto the respective gates.

Our methodology is implemented through a combination of two SAT engines which work in tandem in an interleaved fashion. The first engine is an inexpensive, DPLL based engine designed to catch most of the "easy to prove" equivalences in the vicinity of the equivalence cut. The second engine uses a more advanced general purpose SAT solver (in our case the GRASP [MSS99] solver) to deduce the relatively more difficult equivalences. The two engines are described below.

5.4.1 Segment sweeping based deduction

This engine is designed to catch all equivalence pairs (x, x') such that the combined support of x and x' in terms of the current equivalence cut, δ is less than some specified parameter k. The intuition behind this engine is similar to motivation of the node hashing scheme [KK97]. It is roughly analogous to building BDDs, in terms of the current cut, of all those nodes whose support size (in terms of δ) is less than k and deducing all equivalences that can be deduced from these BDDs. However, for reasons discussed in Section 5.3 our method is much more powerful than either the BDD schemes or hashing, even if the hashing is generalized on the lines of [KGP01].

Let δ denote the cutpoint frontier. A *segment* β of this cut is a subset of cutpoints (as well as their equivalent counterparts) of δ. Let *Base*(β) denote all those gates (variables), in the transitive fanout of the cut δ in circuits C_1 and C_2, whose support in terms of δ is a subset of β. Consider a pair of variables $x, x' \in$ *Base*(β). The equivalence deduction procedure is based on the following principle. Under each assignment to the variable of β, each variable of *Base*(β) takes a Boolean value (0 or 1). If x and x' assume the same Boolean value under each of these $2^{|\beta|}$ assignments, then $x \equiv x'$ *globally*, *i.e.* in terms of the primary inputs. Similarly, if $x \not\equiv x'$ (globally) then the following result follows.

Proposition 5.1 *Given segment β of cut δ and two variables $x, x' \in Base(\beta)$, if $x \not\equiv x'$ (globally) then there exists a Boolean value assignment A_β to the variables of β such that $x|_{A_\beta} \neq x'|_{A_\beta}$.*

In fact this proposition can be further strengthened as follows.

Proposition 5.2 *Given segment β of cut δ and two variables $x, x' \in Base(\beta)$, if $x \not\equiv x'$ (globally) then there exists a Boolean value assignment A_β to the variables of β and an assignment A_I to the primary inputs I such that $\beta|_{A_I} = A_\beta$ and $x|_{A_\beta} \neq x'|_{A_\beta}$.*

Based on Proposition 5.1 equivalence relationships are deduced by constructing and manipulating *equivalence classes* as follows. Given a segment β of cut δ the variables $Base(\beta)$ are first put into a single class, Γ. Then each of the $2^{|\beta|}$ assignments to β is explored one by one with the associated values of the variables $Base(\beta)$ under each assignment, using Boolean value propagation through the circuits. Suppose under the first assignment to β, variables Γ_1 evaluate to 1 whereas variables Γ_0 evaluate to 0, where $\Gamma_0, \Gamma_1 \subseteq \Gamma$, $\Gamma_0 \cup \Gamma_1 = \Gamma$ and $\Gamma_0 \cap \Gamma_1 = \emptyset$. Then the class Γ is split into two sub-classes, Γ_0 and Γ_1. This process is repeated for each current class, after each assignment (and value propagation) to β. After exploring all $2^{|\beta|}$ assignments to the segment, if two variables x and x' lie in a common class, it follows from Proposition 5.1 that $x \equiv x'$ holds globally.

Using the result of Proposition 5.2 the above strategy can be improved considerably. First, when branching on the segment variables (*i.e.* exploring the $2^{|\beta|}$ possible assignments to segment variables) complete Boolean value propagation is done after each variable assignment. The propagation is carried both in front of and behind the cut, using the functional gate level circuit description as well as all previously deduced equivalence relationships. The current branch is terminated as soon a *conflict* is encountered. Secondly, the classes are split if and only if the branching doesn't terminate in a conflict. The above deduction procedure based on branching on a single segment and splitting classes is called a *segment deduction run.*

Given the current cut, δ, the local deduction of equivalences is accomplished through a sequence of segment deduction runs, each with a new segment β drawn from the current cut δ. At the end of these runs we can informally guarantee that all equivalences, deducible from the current cut and within a certain neighborhood of it, have been deduced. In our experiments a segment size of 5 provided a good compromise between deduction power and efficiency.

5.4.2 Global hypothesis based deduction

While the above procedure works fairly well in regions of the circuit where equivalences are abundant and densely scattered, it cannot be generalized to handle all equivalences for the following reasons:

- The distribution of equivalences is highly non-uniform for difficult verification instances. Hence it is impossible to determine a good value of the segment size a priori, in the absence of which the algorithm does a lot of wasted work.
- In some cases, especially arithmetic circuits, missing even a few equivalences can make an appreciable difference in the difficulty of the remaining sub-problem.

We concur with the view of [BS98] on the issue that for more difficult equivalences one needs a robust approach to generate *candidate pairs* of cut-points to verify (we refer to these as *global hypotheses*) and a robust mechanism for verifying these pairs that does not work off a preset hard threshold on the amount of effort to invest in verifying a particular pair.

Our framework for global hypothesis generation and proving, draws on techniques in [Bra93, BS98, vE97] and is similar to the one used in [BS98]. The key difference is that a single SAT algorithm is used both for proving equivalent pairs as well as identifying true negatives[3]. The algorithm pseudo code is shown in Algorithm 5.1.

The **GenerateInitGlobalHypothesisClasses** routine picks up all nodes in the transitive fanout of the current equivalence cut and clusters them into *Global Hypothesis Classes* by running 32-bit parallel simulation on the circuit. Nodes with identical signatures under this simulation lie in the same global hypothesis class. The simulation can be performed with purely random vectors or any other "interesting" set of vectors. This function is used only when global equivalence deduction is invoked for the first time.

ChooseHypothesis selects a pair of nodes, x_1, x_2, belonging to the two circuits from a global hypothesis class, such that the pair is topologically closest to the current equivalence cut. This hypothesis is resolved by invoking a SAT solver on the formula denoting $x_1 \oplus x_2$. All previously deduced equivalences are part of this formula. If the formula is unsatisfiable then $x_1 \equiv x_2$. Thus all previously deduced equivalences currently tagged

[3][BS98] used BDDs for the proving equivalences and heuristically combined it with a randomized SAT algorithm implemented on the BDDs to identify some of the true negatives.

Algorithm 5.1 Global Hypothesis Deduction

```
GenerateInitGlobalHypothesisClasses
while Outputs Unresolved & Not Deduced Sufficient New Equivs. do
  $(x_1, x_2)$ = ChooseHypothesis
  Status = ResolveHypothesis$(x_1, x_2)$
  if Status = "TRUE_NEGATIVE" then
    RefineGlobalHypothesisClasses
    if Outputs in different classes then
      return UNEQUAL, test
    end if
  else
    MergeEquivClasses$(x_1, x_2)$
    if Outputs in the same EquivClass then
      return EQUAL
    end if
  end if
end while
```

onto x_1 and x_2 are merged (routine **MergeEquivClasses**). If the formula is satisfiable, the solution returned provides a vector to simulate and refine *i.e.* to split the current global hypothesis classes (routine **RefineGlobalHypothesisClasses**).

This process iterates until the primary outputs are resolved or a certain number (a parameter to the algorithm) of new cutpoints are deduced. If the primary outputs are deduced as inequivalent, the algorithm returns a witness vector (**test** in Algorithm 5.1) under which the outputs assume different values.

5.4.3 Overall Algorithm

The overall algorithm alternately invokes segment sweeping based deduction and global hypothesis based deduction, switching from one to the other using the following heuristic. The algorithm is initiated by applying segment sweeping on the initial cut, which is the set of common primary inputs I for the two circuits. If the segment sweeping runs from the current cut yield new equivalences, the equivalence cut is advanced by incorporating the new equivalences and segment sweeping is reinvoked on this new cut. This process is iterated until segment sweeping from the current cut reveals no new equivalences.

At this stage the algorithm switches to global hypothesis based deduction, resolving candidate pairs until the primary outputs are resolved or a pre-specified number of new cutpoints are deduced. The cut is advanced to include the new equivalences and segment sweeping is again invoked from the new cut.

Note that the global hypothesis classes maintained and manipulated by the global hypothesis deduction engine are different from the equivalence classes created and subsequently discarded in each segment deduction run. New equivalences deduced through segment sweeping are removed from the global hypothesis classes. Thus, equivalences deduced by segment sweeping are never taken up for resolution by global hypothesis deduction.

5.5 Experimental Results

This section presents experimental results based on a preliminary implementation of the proposed methodology for SAT based CEC. It has been implemented in C and uses the **GRASP** SAT solver [MSS99] for the global hypothesis based deduction phase (Section 5.4.2). Our experiments were run on a Sun Ultra Sparc-1 with 256 Mbytes of memory. The current interface to **GRASP** is through files[4] but the reported runtimes do not include the file I/O times since this can be removed easily through a better integration of the tools. As mentioned earlier, the main objective is to present a realistic assessment of a SAT based CEC tool, rather than to present an optimized and complete equivalence checker.

We present two sets of results. The first compares our method against four tools which in our opinion represent the state of the art in SAT based combinational equivalence checking.

1. **RL_GRASP [MSG99] :** An implementation of **GRASP** augmented with *Recursive Learning* [KS97].

2. **RL_CGRASP [MSeS99] :** An implementation of **GRASP** augmented with Recursive Learning as well a framework for exploiting circuit topology.

3. **HANNIBAL [Kun93] :** A CEC tool using Recursive Learning and a test generator.

[4]**GRASP** is written in C++ whereas our tool is in C.

Circuit	RL_GRASP (secs)	RL_CGRASP (secs)	HANNIBAL (secs)	Implication Graph (secs)	Our Method (secs)
C432	2.8	3.6	3	1.3	0.7
C499	6.8	8.8	6	1.4	1.17
C1355	18.0	27.4	19	7.0	2.37
C1908	94.8	153.0	26	19.5	3.87
C2670	56.4	74.6	231	24.1	4.46
C3540	4006	2560	2057	791.0	38.94
C5315	445.4	476.6	797	33.4	6.96
C6288	109.6	43.6	48	8.9	5.04
C7552	2124	2868	4724	570.1	23.11

Table 5.1: Verifying original vs. irredundant circuits

4. **Implication Graph based method [TGH97]:** Presents a tuned and optimized implementation of a backtracking SAT algorithm that employs some elements of non-local implications and recursive learning. The algorithms are implemented on a specialized data-structure called the *implication graph.*

The comparative results presented in Table 5.1 show the results of verifying the ISCAS'85 benchmarks against their irredundant versions. These benchmarks are relatively easy instances of combinational verification. They are also the only common set of benchmarks on which the above tools have reported results.The results of **RL_CGRASP** were obtained by running the publicly available version of the tool on our machine, using optimized settings which the authors [MSeS99] kindly provided. Since **RL_GRASP** was not available we used the runtime numbers from [MSG99] appropriately scaling for architectural differences. The results for **HANNIBAL** are the original numbers from the paper [Kun93] since we were unable to find suitable scaling data for the Sparc ELC station used by the authors in those experiments. The results of [TGH97] are the numbers from the original paper reported on a DEC Alpha Station $250^{4/266}$, which is a machine comparable in performance to our own. Although, a direct exact value to value comparison is neither fair nor intended, the results of Table 5.1 clearly demonstrate that our method consistently outperforms all the other techniques. Especially noteworthy is the fact that on the three hardest instances of the set, namely C5315, C3540 and C7552 our method outperforms all the other methods by over two orders of magnitude.

The second set of results provide a comparison on a much more difficult set

Circuit	Mixed Engine [BS98] (secs)	Our Method (secs)
C432	–	2.14
C499	–	0.92
C1355	–	1.1
C1908	–	5.90
C2670	3.5	4.93
C3540	**25.7**	**20.98**
C5315	5.3	27.45
C6288	12.1	14.52
C7552	12.7	35.18

Table 5.2: Verifying original vs. optimized circuits

of instances with a state-of-the-art BDD based mixed engine combinational equivalence checker [BS98]. Table 5.2 reports results on verifying some of the MCNC91 circuits against a version optimized by a general purpose logic optimization script, `script.rugged` from SIS [S+92]. The results of [BS98] are reported on the same machine as ours. It is noteworthy that even with our current untuned and prototype implementation our runtimes are mostly comparable to that of [BS98], sometimes a factor of 2-3 slower. However C3540 is an example where our algorithm is faster. Interestingly enough, this is an example with a fairly non-uniform distribution of cut-points, some of which are fairly hard to deduce. Although we believe the runtime discrepancy can be easily made up and in fact bettered by the improvements listed in Section 5.6, we chose not to produce a commercial tool, but to pursue other directions of research.

5.6 Conclusions and Future Directions

We revisited the application of Satisfiability (SAT) algorithms to CEC and argued the case for SAT as a more robust and flexible engine of Boolean reasoning than BDDs. We presented a simple framework for SAT-based CEC and reported results on a preliminary implementation. The results show a speedup of up to two orders of magnitude compared to state-of-the-art SAT-based methods for CEC. They also demonstrate that this simple algorithm and untuned prototype implementation is only moderately slower and sometimes faster than a state-of-the-art BDD-based mixed-engine commercial CEC tool.

There are several avenues for improvement of the current algorithm and implementation:

- **Variable ordering in the SAT solver:** It is well known that variable ordering can affect the performance of SAT solvers, tremendously[5]. Currently, we have experimented with only a few *static variable ordering* schemes with GRASP. More experimentation in this direction could provide substantial speedups.

- **Better implementation of our CEC framework:** The current data-structure and routines are designed for flexibility of rapid algorithm prototyping rather than optimality of the specific proposed framework. Once rewritten and tuned for efficiency, these could easily speed up the time spent outside the calls to the SAT solver at least by a factor of 2-5. This time contributes 30-70% of the overall reported time.

- **More effective use of Initial Vector Simulation:** Currently the 32-bit parallel vector simulation, used for pruning the hypothesis set, works with randomly generated vectors. However, simulating a more intelligent set of vectors could substantially decrease the number of calls to the SAT solver and boost performance proportionally. One idea to do this is to make use of test vectors that are routinely generated for simulating designs, during the design process. These could be "interesting" vectors proposed by the designer or the ATPG test set for one or both of the circuits being checked.

- **Sharing effort between individual hypothesis checks:** One of the reasons for the efficiency of BDD based methods is their ability to re-use previous work by storing part of the Boolean search space in the form of the BDD itself. While our current method makes use of previously done work by storing and using deduced equivalences as shallow witnesses of conflicts (Section 5.3), a more direct sharing of effort between individual SAT calls, somewhat along the lines of [KMSS00a] could substantially improve performance.

- **Using an improved SAT solver:** As research in SAT solvers produces better algorithms and implementations, performance of the proposed framework will improve. The Chaff SAT solver [MMZ+01] proposed recently has been shown to outperform

[5]just as BDD variable ordering affects the size of BDDs

most other solvers (including **GRASP**) by up to 1-2 orders of magnitude. Integrating such a SAT solver into our framework should certainly enhance the performance of our prototype.

SAT-based CEC methods merit further research and application based tuning before they can surpass almost a decade of research in BDD-based combinational verification. The work presented in this chapter is an attempt to demonstrate the advantageous features and the immense potential of SAT methods in a practical verification setting.

Chapter 6

Using Problem Symmetry to Reduce Search

Recent years have seen significant improvements in SAT solver technology [MSS99, MMZ$^+$01, Zha97]. Much of the success of SAT solvers in EDA can be attributed to these improvements. Almost all current leading edge SAT solvers use a backtracking algorithm based on the classical *Davis-Putnam-Logemann-Loveland procedure (DPLL)* [DLL62] enhanced with some form of non-chronological backtracking and conflict based learning [MSS99, MMZ$^+$01]. Efficient implementations of powerful search pruning techniques such as Boolean constraint propagation, non-chronological backtracking and conflict based learning form the computational backbone of most popular SAT solvers [MSS99, Zha97, BS97, MMZ$^+$01]. This chapter introduces and develops the notion of *problem symmetry* to formally characterize and enhance the search space pruning of such a SAT solver.

The notion of *problem symmetry* stems from the simple observation that in certain regions of the Boolean space the unsatisfiability of the given CNF can be established without using a certain variable, say x. In other words, in this sub-space the CNF is symmetric with respect to x (or this is a *symmetric subspace with respect to* x)[1]. In the context of a backtracking based SAT algorithm this can be used as follows. Consider the backtracking search tree shown in Figure 6.1. When exploring the left branch of branching variable x ($x = 0$) the algorithm computes an (under) approximation of the symmetric sub-space (out of the space explored under the branch $x = 0$) with respect to x (sub-space Θ_1 in Figure 6.1)

[1]Note that this notion of symmetry is distinct from the often used notion of a Boolean function being symmetric with respect to certain variables

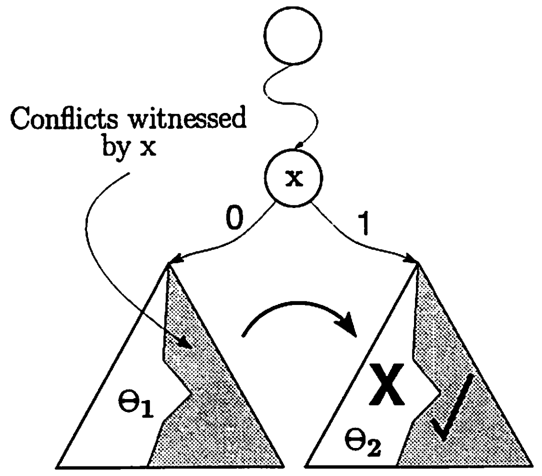


Figure 6.1: Illustration of Symmetry in Search

and in the right branch of x ($x = 1$) the counter-part of this symmetric sub-space (sub-space Θ_2 in Figure 6.1) is pruned.

In this chapter we introduce and develop the notion of *problem symmetry* in search based SAT algorithms. Further, we introduce the notion of essential points and use it to develop a formal characterization of the potential search space pruning that can be realized by exploiting problem symmetry. We show that many popular search pruning techniques such as the pure-literal rule, non-chronological backtracking and conflict based learning that are employed in leading-edge SAT solvers are in fact special cases of pruning under the general theory of essential points. Thereby this work unifies these apparently disparate techniques under a single framework and paves the way for discovering several new pruning techniques. We also propose a new, simple and efficient pruning technique called the *supercubing rule*, based on problem symmetry. Preliminary experimental results demonstrate this to be effective in providing search space pruning over and above the pruning afforded by existing techniques in SAT solvers.

This chapter is organized as follows. Section 6.1 presents some basic definitions and notation used in the exposition. Section 6.2 illustrates the notion of problem symmetry with a few examples. The theory of *essential points* and a formal characterization of problem symmetry is developed in Section 6.3. Section 6.4 presents theoretical results showing several popular pruning techniques used in SAT solvers to be special cases of the general theory of essential points. In Section 6.5 we develop the *supercubing rule*. This is also a special case of problem symmetry based pruning but subsumes some existing pruning techniques and is orthogonal to others. Section 6.6 presents preliminary experimental results validating the efficacy of this rule. Conclusions and directions for future developments are presented in Section 6.7.

6.1 Definitions & Notation

The following discussion is with respect to SAT instances expressed as *conjunctive normal form (CNF)* formulas. Further, the underlying SAT algorithm used in the discussion will be the basic DPLL [DLL62] algorithm, augmented with some form of *conflict analysis*, *non-chronological backtracking* and *conflict clause recording* [MSS99]. As discussed earlier, this is representative of the SAT methods implemented in most leading edge SAT solvers [MSS99, MMZ+01, Zha97].

The exposition will be based on the notational framework developed in Chapter 2 as well as the following. Let l denote a literal of one of the variables $\boldsymbol{V}$. $lit(x)$ refers to a literal of variable x *i.e.* $lit(x)$ is either x or $\overline{x}$. ψ refers to a *minterm* or point in the 2^n Boolean space of variables $x_1, x_2, \ldots, x_n$. Note that a minterm ψ is a complete Boolean assignment to the variables $\boldsymbol{V}$. Further, a formula ϕ can be evaluated under this assignment. In the following we will occasionally use a literal of a variable to refer to a particular value assignment to the variable (*e.g.* $(x = 0) \equiv \overline{x}$) and a cube (minterm) to refer to a partial(complete) value assignment to variables of $\boldsymbol{V}$. $A(x)$ refers to the current assignment of variable x or alternatively the literal corresponding to that assignment.

From Section 2.4.1 $\omega_{\mathcal{I}}(l)$ refers to the clause that was used to imply or deduce the variable x. Extending this terminology, $\mathcal{I}(\omega)$ will be used to denote the set of variables whose literals appear in clause ω and have been assigned values through BCP implications from ω or other clauses. $\mathcal{D}(\omega)$ will denote the set of variables whose literals appear in ω and have been assigned values through decision assignments.

As discussed in Section 2.6, a given conflict condition $\mathcal{X}$ may have several conflict clauses that can be deduced from it. The particular clause or clauses deduced by a given SAT algorithm depend on the specific version of the conflict analysis procedure implemented by it. For the sake of concreteness we will use the following definition of $A_{\mathcal{R}}(\mathcal{X})$ in the sequel. Consider the following recursive marking function $\mathcal{M}(\omega)$, which operates on a clause ω and is defined as

$$\mathcal{M}(\omega) = \mathcal{D}(\omega) \cup \mathcal{I}(\omega) \bigcup_{x \in \mathcal{I}(\omega)} \mathcal{M}(\omega_{\mathcal{I}}(x)) \tag{6.1}$$

Then $V(\mathcal{X}) = \mathcal{M}(\omega(\mathcal{X}))$. Further $V(\mathcal{X})$ can be split into disjoint subsets $V_{\mathcal{D}}(\mathcal{X})$ and $V_{\mathcal{I}}(\mathcal{X})$ which are respectively the decision and implied variables comprising $V(\mathcal{X})$. The clause $\omega_{\mathcal{R}}(\mathcal{X})$ learned and recorded on conflict $\mathcal{X}$ is defined as:

$$\begin{aligned} A_{\mathcal{R}}(\mathcal{X}) &= \{A(x) | x \in V_{\mathcal{D}}(\mathcal{X})\} \\ \omega_{\mathcal{R}}(\mathcal{X}) &= \bigvee_{l \in A_{\mathcal{R}}(\mathcal{X})} \bar{l} \end{aligned} \tag{6.2}$$

Definition 6.1 (Unsatisfiability cube) *Given a clause ω denote by $\mathcal{U}(\omega)$ the unsatisfiability cube of ω which is the set of minterms (assignments) which unsatisfy ω, e.g. given $V = \{x_1, x_2, x_3\}$ and $\omega = (x_1 + \overline{x_2})$, $\mathcal{U}(C) = \{\overline{x_1}x_2x_3, \overline{x_1}x_2\overline{x_3}\}$.*

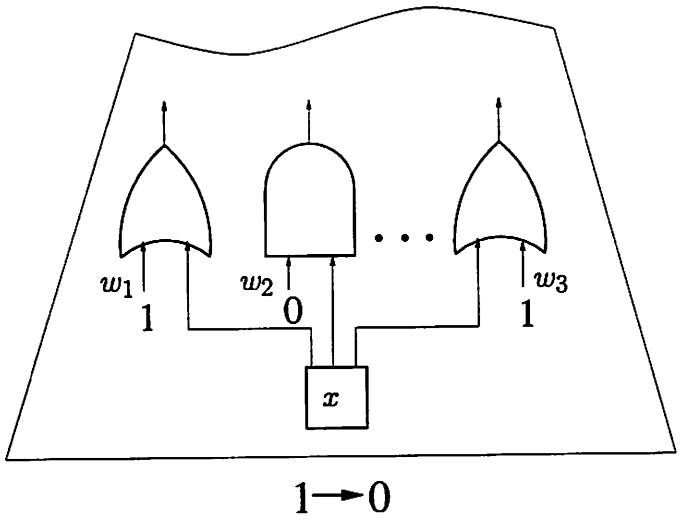
Note that $\mathcal{U}(\omega)$ can also be interpreted as a cube of literals, $\overline{\omega}$. For the above example $\mathcal{U}(\omega) = \overline{x_1}x_2$. In the following we use the two interpretations interchangeably.

6.2 Problem Symmetry in Search

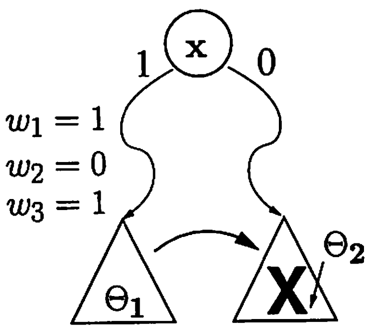
The notion of problem symmetry was introduced and its potential in search space pruning motivated briefly in the beginning of this chapter. In this Section we provide two examples to buttress this understanding and illustrate that 1.) instances of problem symmetry are plentiful in typical SAT instances arising from EDA applications and 2.) current pruning techniques harness only a fraction (albeit inadvertently) of the potential search space pruning afforded by problem symmetry.

Example 6.1 *Consider the sub-circuit shown in Figure 6.2(a). Assume that this is part of a larger circuit on which some SAT problem is being solved*[2]*. Here x is a primary input of the circuit and the three gates shown are the only fanouts of x. Suppose the backtrack tree explored by the SAT algorithm is of the form shown in Figure 6.2(b). Consider the left branch ($x = 1$) of branching variable x. Suppose that under this branch the algorithm subsequently makes the assignments $w_1 = 1, w_2 = 0$ and $w_3 = 1$ (and potentially other*

[2]What is meant here is that an appropriate CNF formula is extracted from the circuit and solved by a SAT solver.



(a) Example Sub-circuit



(b) Backtrack Tree

Figure 6.2: Example of Symmetry in SAT on circuits

assignments as well) and reaches a sub-space Θ_1 (shown in Figure 6.2(b)). Note that in sub-space Θ_1 the value of x is no longer relevant i.e. *the formula is* symmetric with respect to x in Θ_1. *Thus, if the algorithm finds sub-space Θ_1 unsatisfiable then it need not explore the sub-space Θ_2, the counterpart of Θ_1 under the branch $x = 0$, as that too will be unsatisfiable.*

This is a simple and classical case of problem symmetry in SAT instances derived from logic circuits. This case is not explicitly targeted by existing search pruning techniques, so, sometimes such cases may not be effectively covered by existing techniques.

The next example is designed to illustrate that current implementations of conflict clause recording exploit only a fraction of the search space pruning potentially afforded by problem symmetry.

Example 6.2 *Consider the following CNF formula.*

$$\begin{aligned} \phi \;=\; & (\overline{y}+z+w)(\overline{y}+\overline{z}+w)(y+z+w)(y+\overline{z})(\overline{y}+\overline{z}+\overline{w}) \\ & (y+z+\overline{w})(x+\overline{y}+z+\overline{w})(\overline{x}+\overline{y}+z+\overline{w}) \end{aligned} \tag{6.3}$$

A typical backtracking tree for solving this CNF is shown in Figure 6.3. The backtracking algorithm employs conflict analysis, clause recording etc. *The recorded clauses (as per the*

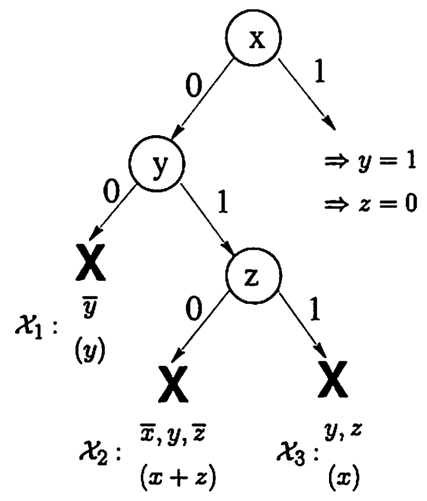

x
0
1
y
⇒ y = 1
⇒ z = 0
0
1
X
$\mathcal{X}_1$: $\overline{y}$ (y)
z
0
1
X
X
$\mathcal{X}_2$: $\overline{x}, y, \overline{z}$ $(x+z)$
$\mathcal{X}_3$: y, z (x)

Figure 6.3: Example of Symmetry in backtrack search

specific scheme described in Section 6.1) are shown below each conflict. Also noted are the set of decision assignments relevant to the conflict, derived through conflict analysis. An analysis of the conflicts in the branch $x = 0$ reveals that x was only relevant in conflict $\mathcal{X}_2$. Alternatively, looking at conflict $\mathcal{X}_2$ it can be concluded that in the sub-space under the branch $x = 0$, x was used in a conflict only when $y = 1$ and $z = 0$. The rest of the sub-space under $x = 0$ represents the symmetric sub-space with respect to x. Thus when exploring the right branch of x, i.e. *$x = 1$ we do not need to explore the sub-space $y = 0 \vee z = 1$. Therefore, on taking the branch $x = 1$ we can immediately assert $y = 1$ and $z = 0$. Note, that the assertion $b = 1$ is also deduced by means of the recorded conflict clause (y). However, derivation of the additional assignment $z = 0$ is made possible only by exploiting problem symmetry more fully through the above analysis.*

Note that the additional pruning realized in the above example is not an artifact of the specific conflict clause recording mechanism used in this work and in this example. Rather, it is a fundamental limitation of any practical implementation of conflict based learning. Conflict based learning techniques typically record a small number (usually just one) of implicates on each conflict. These recorded clauses represent only a fraction of the implicates that could potentially be learned from each conflict. It is neither feasible nor practical to learn all possible implicates. Intuitively, techniques based on problem symmetry, such as the one illustrated above in Example 6.2, attempt to work in the pruning space of such "missed implicates" and are therefore relevant irrespective of the particular clause recording technique used. The *Supercubing technique* presented later in Section 6.5 is motivated by the same reason.

Note that during the search, certain variables, initially picked as decision variables, become deduced variables due to BCP implications from newly added conflict clauses, *e.g.* in Figure 6.3, $y = 1$ can be treated as a deduced assignment implied from the clause (y) recorded on conflict $\mathcal{X}_1$. Such assignments are called *failure-driven assertions (FDA)* [MSS99]. However, $y = 1$ may as well be treated as a decision assignment. In our treatment, FDAs are treated as deduced assignments for the purpose of generating the recorded conflict clauses $\omega_{\mathcal{R}}(\mathcal{X})$. However, for generating the responsible assignments shown in Figure 6.3 (and for the supercubing rule presented in Section 6.5) FDAs are treated as decision assignments. Both versions of the analysis still use Equations 6.1 and 6.2 but generate different sets $V_{\mathcal{D}}(\mathcal{X})$ (and $A_{\mathcal{D}}(\mathcal{X})$).

6.3 The Theory of Essential Points

In this section we develop the notion of *essential points* to formally characterize the search space pruning that can be realized by exploiting problem symmetry.

Definition 6.2 (Essential point) *Given a literal l, a minterm μ is called an $l-$essential point if all clauses of ϕ unsatisfied by the assignment μ (must be at least one[3]) contain literal l, e.g. given $\phi = (\overline{x}+\overline{y})(z)(x+\overline{y}+\overline{z})(\overline{x}+\overline{z})(x+y+\overline{z})$ the minterm xyz is an $\overline{x}$-essential point.*

Definition 6.3 (Symmetric points) *Let μ and μ^* be two minterms in the 2^N Boolean space. μ^* is said to be x-symmetric to μ if it is obtained from μ by inverting the value of variable x in μ, e.g. minterms $\mu = \overline{x}yzw$ and $\mu^* = \overline{x}y\overline{z}w$ are z-symmetric with respect to each other.*

Proposition 6.1 *Let μ be a complete assignment to variables $x_1, x_2, \ldots, x_N$ (i.e. a minterm of 2^N) which satisfies ϕ. Then assignment μ^* which is x-symmetric to μ is either a lit(x)-essential point (where lit(x) $\in \mu$) or a satisfying assignment of ϕ.*

[3]Thus, satisfying assignments of ϕ are not essential points.

Proof: Suppose μ^* is neither a solution nor lit(x)-essential (where $lit(x) \in \mu$). Then there exists a clause ω of ϕ such that μ^* unsatisfies ω and ω does not contain **any** x literal. But then ω is unsatisfied by μ as well. Therefore μ is not a solution of ϕ. Contradiction ! ■

Proposition 6.2 *If assignment μ is lit(x)-essential then assignment μ^*, x-symmetric to μ, is either $\overline{lit(x)}$-essential or is a solution*[4].

Proof: Suppose μ^* is neither a solution nor $\overline{lit(x)}$-essential. Then there exists a clause ω of f such that μ^* unsatisfies ω and ω does not contain **any** x literal. But then ω is unsatisfied by μ as well. Therefore μ is not lit(x)-essential. Contradiction ! ■

For a literal l, the set of l-essential points with respect to the current CNF is denoted by $\mathcal{E}(l)$. The subset of $\mathcal{E}(l)$ lying in a sub-space Θ is denoted by $\mathcal{E}_\Theta(l)$ and by $\mathcal{E}_{sub}(l)$ when the sub-space being referred to is clear from the context.

The search space pruning that can be achieved using the notion of essential points can be operationally defined by the following theorem.

Theorem 6.1 *Suppose the algorithm has explored the left branch of variable x (without loss of generality $x = 0$) and found no solution. Moreover, suppose the algorithm has computed $\mathcal{E}_{\bar{x}}(x)$. Then under the branch $x = 1$ solutions of ϕ must lie in the set of points x-symmetric to points in $\mathcal{E}_{\bar{x}}(x)$ (denoted by $\mathcal{E}^*_{\bar{x}}(x)$).*

Proof: For correctness, the algorithm only needs to ensure that it does not skip any solutions of the CNF in the branch $x = 1$ (it can prune everything else). By Proposition 6.1 solutions can only be points x-symmetric to points in $\mathcal{E}_{\bar{x}}(x)$. ■

Theorem 6.1 implies that for testing satisfiability of ϕ when exploring the branch $x = 1$ the algorithm only needs to explore the set of points $\mathcal{E}^*_{\bar{x}}(x)$. It is also easy to see that it is not necessary to compute the set $\mathcal{E}_{\bar{x}}(x)$ exactly. Any over-approximation of it would work as well, though the amount of pruning would be reduced proportionally.

Under a conflict clause recording scenario, *i.e.* when the algorithm progressively adds implicates of the CNF to the clause database the set of essential points $\mathcal{E}(l)$ for each literal l either remains unchanged or shrinks.

Theorem 6.2 *Let CNF ϕ^+ be obtained from ϕ by adding clause ω^+ to ϕ where ω^+ is an implicate of ϕ. Then, for any literal l, the set of essential points of l in ϕ^+, denoted $\mathcal{E}^+(l)$ must satisfy $\mathcal{E}^+(l) \subseteq \mathcal{E}(l)$.*

[4]In other words a satisfying assignment of ϕ

Proof: Consider any minterm $\mu \notin \mathcal{E}(l)$. Then, there must exist a clause ω of ϕ such that $l \notin \omega$ and $\mu \in \mathcal{U}(\omega)$. But, since $\phi^+ = \phi \cdot \omega^+$, ω is also a clause of ϕ^+. Thus, $\mu \notin \mathcal{E}^+(l)$. Therefore, $\mu \notin \mathcal{E}(l) \Rightarrow \mu \notin \mathcal{E}^+(l)$. ∎

The relevance of Theorem 6.2 is that under a clause recording scenario, when a new clause is added, all partial sets of essential points computed up to that point continue to be valid with respect to the new CNF[5].

6.4 Popular Pruning Techniques: Special Cases of Essential Point Pruning

In the following we show that several popular search pruning techniques such as the *pure-literal rule* [DP60], *non-chronological backtracking (NCB)* and *conflict clause recording* [MSS99] are special cases of the pruning afforded by the theory of essential points. This unifies these techniques under a single framework and paves the way for developing potentially more powerful variants of problem symmetry based pruning.

6.4.1 The Pure-Literal Rule

The Pure-Literal rule [DP60], reviewed in Section 2.4.2, can be used to effect pruning by looking for variables that appear in only one polarity (the *pure polarity*) in unresolved clauses, at the current point in the search, and then asserting the variable to the pure polarity. In effect this means pruning the other branch of the variable. If no solution is found in the explored pure-branch, the pruning effected by the pure-literal rule can be explained by essential points as follows.

The pure-polarity branch of the variable (say $x = 0$) can be considered the left branch of x, which the algorithm explored and found no solution. The other polarity branch $x = 1$ which was pruned by the pure-literal rule is the potential right branch. Thus, if we can prove that the sub-space under the pure-branch $x = 0$ does not contain any x-essential points then the pruning done by the pure-literal rule is explained by Theorem 6.1. In this case the pure-literal rule can be claimed to be a special case of essential point based pruning.

Note that it is sufficient to consider the case when the pure-literal branch of the pure-literal variable is unsatisfiable because in the case when there is a solution under the

[5]However, it can potentially overestimate the essential points with respect to the new CNF with the added clause.

pure-literal branch the algorithm terminates. In such a case the claim of pruning the other branch has no meaning.

Theorem 6.3 *The sub-space under the pure-polarity branch (say $x = 0$) of a pure-literal variable x cannot contain any x-essential points.*

Proof: Consider exploring the pure polarity branch (say $x = 0$) of the pure-literal variable x. By assumption, there is no solution under this branch. Now consider the following algorithm which just explores the sub-space under this branch using a stripped-down DPLL procedure (*i.e.* no BCP or pure-literal rule).

Such an algorithm would simply explore the entire sub-space under the $x = 0$ branch, stopping and *chronologically* backtracking every time the current assignment unsatisfies a clause of the CNF. Let the set of such conflict clauses encountered while exploring this branch be $\omega_1, \omega_2, \ldots, \omega_p$. It is easily seen that $\mathcal{U}(\omega_1) \cup \mathcal{U}(\omega_2) \cup \ldots \cup \mathcal{U}(\omega_p)$ subsumes the sub-space explored under the $x = 0$ branch. Additionally, none of these clauses contain variable x since a conflict clause has all literals unsatisfied by the current assignment and the pure-literal assignment $x = 0$ merely satisfies some clauses and restricts[6] none. The result follows. ∎

6.4.2 Non-Chronological Backtracking (NCB)

As discussed in Section 2.6.2, the notion of *non-chronological backtracking (NCB)* is used to prune areas of the search space by backtracking to the last variable responsible for the current conflict, rather than the last variable in the current assignment stack. This method effects pruning by skipping the right branch of some of the stack variables. Operationally, this is accomplished by deducing an implicate (through conflict analysis) whose unsatisfiability cube subsumes the regions to be pruned.

Another way of looking at this pruning is that NCB prunes the right branch of a variable x, if and only if all conflicts in the left branch of x were independent of (symmetric in) x. This is obviously a special case of symmetry (described by the theory of essential points) which targets pruning sub-spaces symmetric in a particular variable. Before proving this, we state a few simple facts to formalize the operational definition of NCB. These observations follow from the operational definition and correctness of the NCB procedure as developed in [MS95] and reviewed in this dissertation in Section 2.6.2.

[6]An assignment which sets one more literals in a clause to 0 is said to restrict that clause.

- **Fact 1:** NCB pruning is done in a setting where conflict analysis is used to produce conflict clauses (implicates) responsible for the conflict[7].
- **Fact 2:** The deduction procedure for a conflict clause may be simulated by a tree of resolution steps where the leaf clauses are clauses of the original CNF (or previously added conflict clauses) and the variable being resolved out at a node is a deduced variable.
- **Fact 3:** NCB to prune the right branch of variable x happens only on deducing a conflict clause which does not contain any literal of x and whose unsatisfiability cube subsumes the subspace being pruned under the right branch of x.

Proposition 6.3 *If clause ω is the resolvent produced by resolving clauses ω_1 and ω_2 in some common variable (say x) then $\mathcal{U}(\omega) \subseteq \mathcal{U}(\omega_1) \cup \mathcal{U}(\omega_2)$.*

Proof: Without loss of generality, let $\omega_1 = \omega_3 \vee x$ and $\omega_2 = \omega_4 \vee \overline{x}$, where ω_3 and ω_4 are some disjunctions of literals. Then $\omega = \omega_3 \vee \omega_4$. Thus,

$$\begin{aligned}\mathcal{U}(\omega) &= \overline{\omega_3} \cdot \overline{\omega_4} \\ &\subseteq \overline{x} \cdot \overline{\omega_3} \cup x \cdot \overline{\omega_4}\end{aligned}$$

■

Theorem 6.4 *If the right branch of a variable x is eligible for pruning under NCB, then the subspace under the left branch of x (without loss of generality $x = 0$) cannot contain any x-essential points.*

Proof: From Fact 3, there must exist an implicate ω, deduced through conflict analysis which does not contain literals x or $\overline{x}$ and which subsumes the subspace under the unexplored right branch, $x = 1$. Since ω does not contain literals of x it must also subsume the sub-space under the left branch $x = 0$. Moreover, from Fact 2 there must exist clauses $\omega_1, \omega_2, \ldots \omega_k$ of the current CNF which form the leaves of the *resolution tree* simulating the deduction of ω. From the recursive application of Proposition 6.3 it follows that $\mathcal{U}(\omega) \subseteq \mathcal{U}(\omega_1) \cup \mathcal{U}(\omega_2) \cup \ldots \cup \mathcal{U}(\omega_k)$. Thus, clauses $\omega_1, \omega_2, \ldots \omega_k$ collectively cover the subspace under the left branch of x. Also, since the resolution could only be done on deduced variables, and x is a decision variable, clauses $\omega_1, \omega_2, \ldots \omega_k$ cannot have variable x. Therefore none of the points covered by them can be x-essential. ■

[7]The deduced conflict clauses may not be added to the clause database.

6.4.3 Conflict Clause Recording

The simplistic intent of Conflict clause recording [MSS99] is to deduce an implicate (through conflict analysis) responsible for the current conflict and add it to the clause database with the aim of avoiding future occurrences of the same conflict.

Although not apparent from the above statement of the notion, the recorded conflict clauses do in fact effect symmetry based pruning. Consider the following situation. In the left branch of variable x, say $x = 0$, a conflict $\mathcal{X}$ occurs on which a conflict clause $\omega_{\mathcal{R}}(\mathcal{X})$ is learned. Now, suppose $\omega_{\mathcal{R}}(\mathcal{X})$ does not contain literal x (it cannot contain $\overline{x}$). Let the set of assignments, preceding x be given by cube c. Let cube $c_1 = c \wedge \overline{x} \wedge \mathcal{U}(\omega_{\mathcal{R}}(\mathcal{X}))$ and cube $c_2 = c \wedge x \wedge \mathcal{U}(\omega_{\mathcal{R}}(\mathcal{X}))$. Note that c_2 is precisely the sub-space potentially prunable by $\omega_{\mathcal{R}}(\mathcal{X})$ in the right branch $x = 1$ of x.

As shown below, the pruning of sub-space c_2 by clause $\omega_{\mathcal{R}}(\mathcal{X})$ can be accounted for by the theory of essential points. Thus, conflict clause based pruning is a special case of essential point based pruning.

Theorem 6.5 *The symmetry based pruning afforded by a recorded conflict clause $\omega_{\mathcal{R}}(\mathcal{X})$ with respect to a variable x is subsumed by the pruning potentially realizable using essential point based pruning (Theorem 6.1).*

Proof: Using the above notation, note that c_1 is the x-symmetric region corresponding to c_2, the region being pruned. By assumption, literal x does not appear in clause $\omega_{\mathcal{R}}(\mathcal{X})$. Thus, $\mathcal{U}(\omega_{\mathcal{R}}(\mathcal{X}))$ cannot include any x-essential points (follows from Definition 6.2 and Definition 6.1). Thus, $\mathcal{E}(x) \cap \mathcal{U}(\omega_{\mathcal{R}}(\mathcal{X})) = \emptyset$. From this it follows that $\mathcal{E}(x) \cap c_1 = \emptyset$. Therefore, by Theorem 6.1 the x-symmetric counterpart of c_1, namely c_2 lies in the region that can be pruned when exploring the right branch, $x = 1$ of x. ∎

It can be shown that the entire pruning potentially accomplished by a recorded clause, subsequent to its recording can be broken down into a series of right-branch prunings like the above situation[8].

Theorem 6.6 *The search space pruning accomplished by a recorded clause $\omega_{\mathcal{R}}$, subsequent to its recording, can be divided into a set of sub-spaces such that each sub-space lies under the right branch of a variable y, where $\omega_{\mathcal{R}}$ was recorded in the left branch of y and y does not appear in $\omega_{\mathcal{R}}$.*

[8]provided the search is organized as a single tree *i.e.* without restarts.

Proof: Consider the stack of assignments at the time of learning clause $\omega_{\mathcal{R}}$. Let $x_1, x_2, \ldots, x_k$ be the *decision* variables in this stack that are currently in their *left* branches. The Boolean sub-space explored by the SAT algorithm subsequent to recording $\omega_{\mathcal{R}}$ is the union of k disjoint sub-spaces $\Theta_1, \Theta_2, \ldots, \Theta_k$, where Θ_i is defined as follows. In the current assignment stack, erase assignments up to but not including variable x_i (*i.e.* all assignments below x_i). Now flip the variable x_i. Variable x_i is now in its right branch. The sub-space lying below this right branch is Θ_i.

Now, every sub-space pruned by $\omega_{\mathcal{R}}$ subsequent to its recording must lie in $\Theta_1 \cup \Theta_2 \cup \ldots \cup \Theta_k$. Further if x_j appears in $\omega_{\mathcal{R}}$, $\omega_{\mathcal{R}}$ cannot prune any sub-space in Θ_i since it would be satisfied in this sub-space. This completes the proof. ■

From Theorem 6.5 and Theorem 6.6 conflict clause recording can be seen to be a special case of essential point pruning.

6.5 Symmetry Based Pruning through Supercubing

In this section we develop a novel pruning rule based on exploiting problem symmetry. This rule is called the *supercubing rule* after the supercube operator defined below, which is the core operation used in implementing it.

Definition 6.4 (Supercubing Operator ($\mathcal{S}$)) *Given two cubes c_1 and c_2 over the 2^n Boolean space, $\mathcal{S}(c_1, c_2)$ computes the smallest cube containing both c_1 and c_2, i.e. the supercube of c_1 and c_2.*

6.5.1 Supercubing Procedure & Pruning

The algorithm maintains a cube called the *supercube* for each decision variable currently on the decision stack. The supercube of variable x (denoted $\mathcal{S}_x$) is initialized to $\emptyset$ when x is first chosen for branching. In the left branch of x (say $x = 0$) $\mathcal{S}_x$ is updated on each conflict $\mathcal{X}$ where $x \in A_{\mathcal{R}}(\mathcal{X})$ ($A_{\mathcal{R}}(\mathcal{X})$ is computed considering FDAs as decision variables) as follows:

$$\mathcal{S}_x = \mathcal{S}(\mathcal{S}_x, c_S) \qquad \text{where} \quad c_S = \bigwedge_{l \in A_{\mathcal{R}}(\mathcal{X})} l \tag{6.4}$$

After the algorithm has explored the left branch $x = 0$ and found no solution, it would have computed some supercube for x, denoted $\mathcal{S}_x^{final}$. Say $\mathcal{S}_x^{final} = \overline{x} \wedge l_1 \wedge l_2 \wedge \ldots \wedge l_k$

Then in the right branch, $x = 1$ we immediately assert $l_1 = TRUE, l_2 = TRUE, \ldots l_k = TRUE$ *i.e.* the region $x \wedge (\overline{l_1} \vee \overline{l_2} \vee \ldots \vee \overline{l_k})$ is pruned.

The asserted assignments are treated as conscious assignments for the purpose of future conflict analysis and supercubing *i.e.* it is as though these variables $v_{l_1}, v_{l_2}, \cdots, v_{l_k}$ were consciously branched on and the branches $\overline{l_1}, \overline{l_2}, \ldots, \overline{l_k}$ were pruned, while the other branches were explored.

The supercubing rule is inspired by the theory of essential points. However, in its current implementation it is not a special case of the theory of essential points. This is because on a given conflict the assignments used for learning a conflict clause are different from the assignments used for supercubing (FDAs are treated differently for these two purposes). In the absence of this discrepancy[9] it is easy to prove that supercubing becomes a special case of essential point pruning.

6.5.2 Proof of Correctness

The proof of correctness of the algorithm requires proving two propositions:

1. Every supercube-based pruning is legal, *i.e.* the pruned space cannot contain a solution.

2. At any time during the algorithm the following property holds for each minterm, ψ in the Boolean sub-space that the algorithm has already explored (and found unsatisfiable).

Definition 6.5 *Point ψ satisfies Property $\mathcal{P}$ if there exists a cube c_ψ such that $\psi \subseteq c_\psi$ and c_ψ was processed by supercubing (Equation 6.4) under some previous conflict.*

Proof: Note that the algorithm prunes off (and thus implicitly explores) regions of the Boolean space through two kinds of *pruning events*, namely 1.) regular conflicts and 2.) supercube based pruning.

We prove the above two propositions simultaneously by induction on the sequence of pruning events. The overall idea is to prove that if all points pruned by all previous pruning events satisfy Property $\mathcal{P}$; then

[9]We experimented with this version of supercubing and found it to be ineffective in providing any additional pruning over clause recording.

a.) points pruned by the current pruning event satisfy Property $\mathcal{P}$, and

b.) supercube-based pruning is legal.

Base Case : Since pruning occurs only in the right branch of a variable, the first pruning event must be a conflict and by definition, the algorithm would generate a conflict clause covering the pruned region and do supercubing on it. Therefore, all pruned points satisfy Property $\mathcal{P}$.

Induction hypothesis : Suppose points pruned by the first k pruning events satisfy Property $\mathcal{P}$ and are legal prunings.

Induction proof : Consider the $k+1^{th}$ pruning event. If this is a regular conflict the proof trivially follows as per the base case. So consider the case when it is supercube based pruning performed in the right branch $x = 1$ of some variable x. The region pruned by supercubing $\mathcal{S}_x^{prune} = x \wedge \overline{\exists x \cdot \mathcal{S}_x^{final}}$. Consider any point $\psi^* \in \mathcal{S}_x^{prune}$ and point ψ, which is x-symmetric to ψ^*. Obviously ψ was examined by the algorithm in the left branch of x. Further, $\psi \notin \mathcal{S}_x^{final}$. Also, by the induction hypothesis there exists cube c_S such that $\psi \in c_S$ and c_S was processed by supercubing. Thus, since $c_S \not\subseteq \mathcal{S}_x^{final}$ cube c_S must not have variable x which means that it covers point ψ^* as well. Hence all points in $\mathcal{S}_x^{prune}$ are covered by conflict clauses that have already been discovered and processed by the algorithm. This also means that the current pruning is a legal one (since the pruned space is obviously unsatisfiable).

Note that in reality there is a third kind of pruning event, namely BCP deductions, which prune off regions of the Boolean space. However, this pruned space is **completely** accounted for by the conflict clauses of the conflicts lying below this deduction. A simple way to prove this is to take the current branching tree and "push" all BCP deductions to the leaves of the tree *i.e.* after all the decision assignments in each branch. Since in our procedure all conflict clauses are composed entirely of decision assignments the same conflicts will still occur, but there will be no BCP-pruned areas this time. Here, the conflict clauses can be trivially seen to cover the entire pruned areas. Also we have not considered pure-literal rule based pruning in this proof since this rule is a special case of Supercubing (see Proposition 6.4). ■

6.5.3 Supercubing and Other Pruning Techniques

Proposition 6.4 *The pure-literal rule is a special case of supercubing based pruning.*

Proof: Consider a pure literal variable x. Assume that it is a decision variable whose left branch is the pure-literal branch *i.e.* the assignment dictated by the pure-literal rule, whose right branch has been pruned off. Note that since a pure-literal assignment just satisfies clauses, it can never contribute towards a conflict. This can be seen through a careful examination of our specific definition of conflict analysis as stated in Section 6.1. A pure-literal assignment can never be part of $A_{\mathcal{R}}(\mathcal{X})$[10]. Thus, if x were a decision variable whose left branch $x = 0$ was explored, x would never contribute to any conflict in this sub-space. Thus its computed supercube would remain $\emptyset$. Therefore, supercubing based pruning would dictate pruning the right branch, which is what the pure-literal rule does.

■

From the above proof it follows that in some of the instances where a null supercube is computed for a decision variable x, supercubing based pruning of the right branch of x is synonymous with an application of the pure-literal rule on x. In other such cases (*i.e.* where a null supercube from the left branch causes supercubing to prune the right branch) the behavior of the algorithm is identical to non-chronological backtracking. Thus, supercubing overlaps with some instances of non-chronological backtracking. In fact, we conjecture that supercubing subsumes non-chronological backtracking. All our experiments thus far have not yielded a single case where non-chronological backtracking, implemented in the conventional fashion, could prune a sub-space that supercubing could not. However, the operational definition of NCB given in the literature (and re-stated in Section 6.4.2) is not precise enough to prove or challenge our conjecture. A more unambiguous definition of NCB would probably be required to do this. This could be an interesting problem for future research.

Conjecture: *Non-chronological backtracking is a special case of supercubing based pruning.*

6.6 Experimental Results

This section presents preliminary experimental results validating the efficacy of the supercubing pruning rule described in Section 6.5. The pruning rule has been implemented in a prototype SAT solver modeled on the lines of the GRASP SAT solver [MSS99]. The prototype solver implements all the algorithmic features of GRASP including conflict anal-

[10]This fact would hold for any heuristic to compute the set $A_{\mathcal{R}}(\mathcal{X})$.

Benchmark	Best Order # Nodes		Worst Order # Nodes	
	Orig.	With SC	Orig.	With SC
ssa-0432-003	1371	1050	3316	1074
ssa-2670-130	44039	38812	109766	66142
bf-0432-007	11487	10811	27298	9099
queueinvar8	3211	2983	5842	5842
aim-50-1_6-no-2	27	26	150	84
aim-100-1_6-no-1	120	64	881	455
aim-200-1_6-y-1-4	291	193	1155	354
aim-200-1_6-no-3	457	559	6671	1252
par-16-1-c	6543	6543	6543	6543
hole 6	719	719	817	817

Table 6.1: Experimental results with Supercubing

ysis, non-chronological backtracking, conflict based learning and various ordering heuristics (*e.g.* DLCS, DLIS, MSTS, MSOS etc.). However, the solver has not yet been software engineered for efficiency since its purpose is simply to evaluate the first order efficacy of some pruning techniques.Therefore the reported results are in terms of number of nodes in the SAT search tree, rather than CPU runtimes since reporting the latter would be unfair and not particularly informative.

Preliminary results on some selected SAT benchmarks from the DIMACS suite and bounded model checking [BCCZ99] are reported in Table 6.1. The benchmark examples have been chosen to be representative of the examples that we ran, ranging from the ones where supercubing gave the maximum improvement to ones where it was not so effective.

For each benchmark the solver was run in two configurations with four possible orderings, DLCS, DLIS, MSTS, MSOS[11] (*i.e.* eight configurations in total) 1.) **ORIG:** without supercubing but with NCB and clause recording, and 2.) **With SC:** same as ORIG except supercubing is used also. For each benchmark the best and the worst ORIG results (in terms of number of nodes in the backtracking tree) were chosen and are reported in columns 2 and 4 respectively. The corresponding results **with SC** (*i.e.* with the same ordering heuristic as the ORIG result) are reported in columns 3 and 5 respectively.

As shown in Table 6.1 the search tree size decreases in most cases, sometimes

[11]Refer to the GRASP user manual for details on these heuristics.

quite significantly. In the odd case (in our experience less than 1% of the cases) *e.g.* `aim-200-1_6-no-3` there is a slight increase in the number of nodes. This is because supercubing and associated pruning disturbs the number of recorded clauses and hence the variable order slightly. However, overall supercubing proved beneficial for both the best order and the worst order. The improvements in the case of the worst ordering were more significant suggesting that this pruning technique can partially correct a poor ordering. The supercubing itself added virtually nothing to the runtimes since most of the book-keeping required for it was being done by conflict analysis. The additional supercubing operations were efficiently implemented by bit-vector operations. Thus gains in number of search tree nodes translate directly to runtime gains. Also, since supercubing based pruning partly overlaps with the pruning provided by conflict-based learning using supercubing frequently led to fewer recorded clauses. This feature of supercubing can be used to partly alleviate the clause database memory problems that are becoming an issue in current SAT solvers [MMZ$^+$01].

6.7 Conclusions & Future Directions

In this chapter we introduced and formalized the notion of *problem symmetry* in search based SAT algorithms. We developed the *theory of essential points* to formally characterize the potential search space pruning that can be realized by exploiting problem symmetry. We unified several powerful search pruning techniques used in modern SAT solvers under a single framework, by showing them to be special cases of the theory of essential points. We also proposed a new pruning rule based on problem symmetry and showed it to provide additional search space pruning over the pruning realized by current techniques.

Current SAT solvers integrate fairly sophisticated search pruning techniques in a very tightly and efficiently engineered software framework. However, there is very little fundamental understanding of how these techniques interact, what search space they prune and what the margin for improvement is. This work is a step towards answering these questions. We believe that it is possible to derive a whole family of search pruning techniques with varying cost-power tradeoffs, under the general purview of problem symmetry based pruning. The supercubing rule presented in Section 6.5 is a simple case in point. It is quite obviously a very weak and cheap realization of symmetry based pruning. However, it still improves over the state-of-the-art, demonstrating the potential for improvement. Our current and future research efforts are aimed at realizing some of this potential.

Chapter 7

Conclusions & Future Directions

Recent years have seen dramatic improvements to SAT algorithms and tools and an increased application of SAT methods to solving EDA problems. Despite this progress there is an immense potential for improvement both in the understanding and application of SAT methods in EDA as well in SAT methods themselves. The goal of this dissertation was to advance the theory, practice and core technology of SAT algorithms in the context of EDA applications. To this end the research was organized to address the following three issues:

1. Tight complexity analysis of SAT based EDA applications which accounts for characteristics of real-life problem instances.
2. SAT formulation of EDA applications.
3. Core SAT algorithms and pruning techniques.

These three issues were addressed in the three parts of the dissertation.

The first part of the dissertation was presented in Chapter 4. This chapter presented a worst case complexity analysis for a SAT formulation of the combinational ATPG problem which incorporated salient characteristics of problem instances encountered in real life. Incidentally, this analysis is also one of the first attempts at reconciling the theoretical intractability of combinational ATPG with the relative ease with which practical instances of it are solved. The analysis was based on the SAT formulation of ATPG proposed by Larrabee [Lar92], with a caching based variant of simple backtracking (see Section 4.3) used to model the SAT solver.

Under this model of the algorithm the complexity of ATPG on a given circuit was characterized in terms of a topological property of the circuit, the *undirected circuit cut-width*. Theoretical arguments and experimental results confirmed that this property could be used to predict polynomial runtimes of ATPG for a wide range of practical VLSI circuits.

Specifically, the analysis was used to define a class of circuits called log-bounded-width circuits which were shown to be efficiently testable. Additionally, this class of circuits was shown to subsume the class of k-bounded circuits. Our experiments on a wide range of benchmark and artificially generated circuits showed that they exhibited the log-bounded-width property. On an intuitive level the log-bounded-width property essentially captures the "treeness" of the circuit. As long as a circuit has limited reconvergence (not necessarily local reconvergence), the log-bounded-width property can be expected to apply.

The theme of the second part of the dissertation was to investigate more effective ways of solving EDA problems using existing SAT methods. In Chapter 5 we revisited the application of Satisfiability (SAT) algorithms to the *Combinational Equivalence Checking (CEC)* problem. CEC is an important and well researched EDA problem. Traditionally, BDDs have formed the computational core of CEC tools and previously proposed SAT based solutions were not popular. In Chapter 5 we argued the case for SAT as a more robust and flexible engine of Boolean reasoning for the CEC application than BDDs. We presented a simple framework for SAT based CEC and reported results on a preliminary implementation of this methodology. The results showed a speedup of up to two orders of magnitude compared to state-of-the-art SAT based methods for CEC. Additionally, the relatively simple SAT based CEC approach and its initial prototype implementation proved to be only moderately slower and sometimes faster than a state-of-the-art BDD-based mixed-engine commercial CEC tool.

While SAT based CEC methods may need further research and application based tuning to surpass almost a decade of research in BDD based combinational verification, we showed that SAT based methods for verification are certainly promising and merit continued research. Further, the work demonstrated the specific advantageous features of SAT methods in a practical verification setting.

The third part of the dissertation, presented in Chapter 6, proposed theoretical and practical advancements to the core algorithms and techniques used in modern SAT solvers. Current SAT solvers integrate fairly sophisticated search pruning techniques in a very tightly and efficiently engineered software framework. However, there is little fundamental understanding of how these techniques interact, what search space they prune and what the margin for improvement is. Part of the motivation for the work presented in this Chapter was to address these issues.

Specifically, in this chapter we introduced and formalized the notion of *problem symmetry* in search based SAT algorithms. We developed the *theory of essential points* to formally characterize the potential pruning that can be realized by exploiting problem symmetry. We unified most search pruning techniques used in modern SAT solvers under a single framework, by showing them to be special cases of the theory of essential points. We also proposed a new pruning rule exploiting problem symmetry and demonstrated that it could provide additional search space pruning over the pruning realized by current techniques.

7.1 Future Directions

7.1.1 Input-distribution based complexity analysis of EDA-SAT problems

The analysis presented in Chapter 4 can be developed further. Although the analysis is a significant improvement over traditional worst case complexity analysis, it is based on a rather simplistic model of the ATPG-SAT algorithm. Practical ATPG-SAT engines [SBSV96, Lar92] employ a host of other search pruning techniques to reduce complexity, such as random test pattern generation [ABF95]. In some cases these techniques do bring about substantial reduction in complexity. The benchmark C6288 from the ISCAS85 suite is an example with a large cut-width but is efficiently testable through random test pattern generation. The presented analysis can be made tighter and more realistic by including more algorithmic features of ATPG and SAT tools in the algorithm model.

Secondly, although the analysis of Chapter 4 was aimed at the combinational ATPG problem, the notion of cut-width and its use in analyzing the complexity of a CIRCUIT-SAT based problem formulation is not endemic to the ATPG problem. The same analysis framework could be modified to apply to other EDA problems using a CIRCUIT-SAT formulation.

Finally, a natural extension of cut-width is the notion of *tree-width*. Simply put, tree-width of a graph is an index of the degree to which the topology of the graph resembles a tree. The significance of this measure is that several combinatorial problems, such as graph coloring, which are intractable on general graph structures, have polynomial time solutions on bounded tree-width graphs. Further, in the realm of EDA, several problems, such as *technology mapping, combinational ATPG, etc.* are known to be efficiently solvable on tree-structures but are intractable in the general case. An interesting line of research would be to investigate the tree-width properties of graph structures derived from practical circuits and to reason about the complexity of solving various EDA problems on graphs with small or bounded tree-widths.

7.1.2 Using SAT methods in EDA problems

Section 5.6 details a list of possible enhancements to the SAT based CEC framework proposed in Chapter 5. These would provide an immediate and perhaps commercially interesting area of work for improving SAT based CEC.

Another avenue of future research is to find effective means of having SAT methods co-operate with established engines such as BDDs, structural ATPG methods, simulation *etc.* which have traditionally formed the mainstay of EDA tools. Some initial efforts in this direction have been made in [GA98, BS98, MJT+99, PK00, HSH+00, GYAG00]. However, in many of these works the use of SAT solvers is either very limited or too specific to be applied to other EDA problems. Moreover, some of these works used older SAT solvers which are not representative of the capabilities of current leading-edge SAT solvers. Thus, this avenue of research should be revisited and investigated further. This will also naturally pave the way for the use of SAT solvers in new EDA problems.

Currently the application of SAT methods is largely restricted to problems on combinational logic circuits. The only use of SAT for reasoning on sequential circuits has been *bounded model checking* [BCCZ99]. Since the domain of problems on sequential logic circuits is much richer and more challenging than the combinational domain, research facilitating the use of SAT methods on sequential circuits would greatly widen the gamut of SAT applications in EDA.

7.1.3 Improvements to SAT algorithms and tools

We believe that it is possible to derive a whole family of search pruning techniques with varying cost-power tradeoffs, under the general purview of the theory of essential points, presented in Chapter 6. The supercubing rule presented in Section 6.5 is a simple case in point. It is quite obviously a very weak and cheap realization of essential point pruning. However, it still improves over the state-of-the-art clearly demonstrating the potential for improvement. This would provide an interesting area for future research.

Another area of SAT research is that of incomplete SAT algorithms for EDA applications. Incomplete SAT solvers such as **GSAT** [SLM92] and **WSAT** [SKC96] have been proposed in the Artificial Intelligence community and successfully applied in that domain to solve several problems considered challenging or unsolvable by conventional branch and bound complete SAT solvers. While these particular incomplete SAT solvers have not been successful in solving EDA problems, techniques such as random and weighted random simulation are routinely and successfully used in several testing and verification applications. These techniques are in some sense incomplete, stochastic SAT techniques, albeit very naive ones. We believe that it should be possible to engineer incomplete SAT techniques that inherit the flavor of random simulation and are most aptly suited for EDA applications.

Bibliography

[ABE00] Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic Reachability Analysis Based on SAT-Solvers. In Susanne Graf and Michael Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2000)*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer, March 2000.

[ABF95] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press and John Wiley & Sons Inc., 1995.

[AdS00] Miron Abramovici and José T. de Sousa. A SAT Solver Using Reconfigurable Hardware and Virtual Logic. *Journal of Automated Reasoning*, 24(1-2):5–36, February 2000.

[AMS01] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah. MINCE: A Static Global Variable-Ordering for SAT and BDD. In *International Workshop on Logic and Synthesis (IWLS01)*, June 2001.

[BC00] Per Bjesse and Koen Claessen. SAT based Verification without State Space Traversal. In Warren A. Hunt and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 372–389. Springer, November 2000.

[BCC+99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic Model Checking using SAT procedures instead of

BDDs. In *Proceedings of the* 36^{th} *Design Automation Conference*, pages 317–320, June 1999.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In W. Rance Cleaveland, editor, *Proceedings of the Fifth International Conference on Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, March 1999.

[BCH90] E. Boros, Y. Crama, and P. L. Hammer. Polynomial-time Inference of All Valid Implications for Horn and Related Formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.

[BCHS94] E. Boros, Y. Crama, P. L. Hammer, and M. Saks. A Complexity Index for Satisfiability Problems. *SIAM Journal of Computing*, 23(1):45–49, February 1994.

[BCRZ99] Armin Biere, Edmund Clarke, Richard Raimi, and Yunshan Zhu. Verifying safety properties of a PowerPC microprocessor using symbolic model checking without BDDs. In Nicolas Halbwachs and Doron Peled, editors, *Proceedings of the* 11^{th} *International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 60–71. Springer, July 1999.

[Ber] Daniel Le Berre. SAT Live! `http://www.satlive.org`.

[Ber91] C. Leonard Berman. Circuit Width, Register Allocation and Ordered Binary Decision Diagrams. *IEEE Transactions on CAD*, 10(8):1059–1066, August 1991.

[Ber01] Daniel Le Berre. Exploiting the real power of Unit Propagation Lookahead. In Henry Kautz and Bart Selman, editors, *The LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*, volume 9 of *Electronic Notes in Discrete Mathematics*. Elsevier Science Publishers, June 2001.

[BF85] Franc Brglez and Hideo Fujiwara. A Neural Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *International Symposium on Circuits and Systems*, June 1985.

[BGV01] Randal E. Bryant, Steven German, and Miroslav N. Velev. Processor Verification Using Efficient Reductions of the Logic of Uninterpreted Functions to Propositional Logic. *ACM Transactions on Computational Logic (TOCL)*, 2(1):1–41, January 2001.

[BLM01] Per Bjesse, Tim Leonard, and Abdel Mokkedem. Finding Bugs in an Alpha Microprocessor Using Satisfiability Solvers. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the* 13th *International Conference on Computer Aided Verification (CAV'01)*, volume 2102 of *Lecture Notes in Computer Science*, pages 454–464. Springer, July 2001.

[BMS00] Luís Baptista and João P. Marques-Silva. Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. In Rina Dechter, editor, *Proceedings of the* 6th *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1894 of *Lecture Notes in Computer Science*, pages 489–494. Springer, September 2000.

[Bor97] Arne Borälv. The industrial success of verification tools based on Stålmarck's Method. In Orna Grumberg, editor, *Proceedings of the* 9th *International Conference on Computer Aided Verification (CAV'97)*, volume 1254 of *Lecture Notes in Computer Science*, pages 7–10. Springer, June 1997.

[Bra93] Daniel Brand. Verification of Large Synthesized Designs. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 534–537, Nov 1993.

[BRSVW87] Robert K. Brayton, Richard Rudell, Alberto L. Sangiovanni-Vincentelli, and Albert Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Transactions on CAD*, 6(6):1062–1082, November 1987.

[Bry86] Randal E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C(35):677–691, August 1986.

[BS97] Robert J. Bayardo and Robert C. Schrag. Using CSP Lookback Techniques to Solve Real-World SAT Instances. In *Proceedings of the Fourteenth National*

Conference on Artificial Intelligence (AAAI-97), pages 203–208. AAAI Press, July 1997.

[BS98] Jerry R. Burch and Vigyan Singhal. Tight Integration of Combinational Verification Methods. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 570–576, November 1998.

[BT89] C. Leonard Berman and Louise H. Trevillyan. Functional Comparison of Logic Designs for VLSI Circuits. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 456–459, November 1989.

[CG96] Chih-Ang Chen and Sandeep K. Gupta. A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits. In *Proceedings of the* 33^{rd} *Design Automation Conference*, pages 209–214, June 1996.

[CK99] Pinhong Chen and Kurt Keutzer. Towards True Crosstalk Noise Analysis. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 132–137, November 1999.

[Dec90] Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41(3):273–312, January 1990.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5:394–397, July 1962.

[DMSV88] Srinivas Devadas, H.-K. T. Ma, and Alberto L. Sangiovanni-Vincentelli. Logic Verification, Testing and Their Relationship to Logic Synthesis. In *Testing and Diagnosis of VLSI and ULSI*, pages 181–246. Kluwer Academic Publishers, 1988.

[DP60] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[dSMSA01] José T. de Sousa, João P. Marques-Silva, and Miron Abramovici. A Configurable Hardware/Software Approach to SAT Solving. In *Proceedings of* *the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, April 2001.

[DW83] M. D. Davis and E. J. Weyuker. *Computability, Complexity and Languages.* Academic Press, 1983.

[EC95] Luis A. Entrena and Kwang-Ting Cheng. Combinational and Sequential Logic Optimization by Redundancy Addition and Removal. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 14(7):909–916, July 1995.

[eSMSSS97] Luís Guerra e Silva, João P. Marques-Silva, Luís M. Silveira, and Karem A. Sakallah. Satisfiability Models and Algorithms for Circuit Delay Computation. In *ACM Workshop on Timing Issues in the Specification and Synthesis of Digital Systems (TAU)*, December 1997.

[eSSMS99] Luís Guerra e Silva, Luís M. Silveira, and João P. Marques-Silva. Algorithms for Solving Boolean Satisfiability in Combinational Circuits. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 526–530, March 1999.

[FDK98] Farzan Fallah, Srinivas Devadas, and Kurt Keutzer. Functional Test Generation Using Linear Programming and 3-Satisfiability. In *Proceedings of the* 35^{th} *Design Automation Conference*, pages 528–533, June 1998.

[Fuj88] Hideo Fujiwara. Computational Complexity of Controllability/Observability Problems for Combinaitonal Circuits. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, pages 64–69, June 1988.

[GA98] Aarti Gupta and Pranav Ashar. Integrating a Boolean Satisfiability Checker and BDDs for Combinational Equivalence Checking. In *Proceedings of the* 11^{th} *International Conference on VLSI Design*, pages 222–225, January 1998.

[GB99] Debabrata Ghosh and Franc Brglez. Equivalence Classes of Circuit Mutants for Experimental Design. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 432–435, May 1999.

[GF01] Emil I. Gizdarski and Hideo Fujiwara. A Framework for Low Complexity Static Learning. In *Proceedings of the* 38th *Design Automation Conference*, pages 546–549, June 2001.

[GGYA01] Aarti Gupta, Anubhav Gupta, Zijiang Yang, and Pranav Ashar. Dynamic Detection and Removal of Inactive Clauses in SAT with Application in Image Computation. In *Proceedings of the* 38th *Design Automation Conference*, pages 536–541, June 2001.

[Gin93] Matthew L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[Goe81] Prabhakar Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, C-30(3):215–222, March 1981.

[GPFW97] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the Satisfiability (SAT) Problem: A Survey. In D.-Z. Du, Jun Gu, and P. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Computer Science*, pages 19–151. American Mathematical Society, 1997.

[gre] Greentech Computing Ltd. (UK). `http://www.greentech-computing.co.uk`.

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–437. AAAI Press, July 1998.

[GW99] Jan F. Groote and Joost P. Warners. The propositional formula checker HeerHugo. Technical Report SEN-R9905, CWI, January 1999.

[GYA$^+$01a] Malay Ganai, Praveen Yalagandula, Adnan Aziz, Andreas Kuehlmann, and Vigyan Singhal. SIVA: A System for Coverage Directed State Space Search.

Journal of Electronic Testing: Theory and Applications (JETTA), 17(1):11–27, February 2001. Kluwer Academic Publishers.

[GYA+01b] Aarti Gupta, Zijiang Yang, Pranav Ashar, Lintao Zhang, and Sharad Malik. Partition-based decision heuristics for image computation using sat and bdds. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 531–538, November 2001.

[GYAG00] Aarti Gupta, Zijiang Yang, Pranav Ashar, and Anubhav Gupta. SAT-Based Image Computation with Application in Reachability Analysis. In Warren A. Hunt and Steven D. Johnson, editors, *Proceedings of the 3^{rd} International Conference on Formal Methods in Computer Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 354–371. Springer, November 2000.

[HGRC98] Michael Hutton, J. P. Grossman, Jonathan Rose, and Derek Corneil. Characterization and paramterized generation of synthetic combinational benchmark circuits. *IEEE Transactions on CAD*, 17(10):985–996, October 1998.

[Hoc97] Dorit S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1997.

[HSH+00] Pei-Hsin Ho, Thomas Shiple, Kevin Harer, James Kukula, Robert Damiano, Valeria Bertacco, Jerry Taylor, and Jiang Long. Smart Simulation Using Collaborative Formal and Simulation Engines. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 120–126, November 2000.

[IS75] Oscar H. Ibarra and Sartaj K. Sahni. Polynomially Complete Fault Detection Problems. *IEEE Transactions on Computers*, C-24(3):242–249, March 1975.

[JMF95] Jawahar Jain, Rajarshi Mukherjee, and Masahiro Fujita. Advanced Verification Techniques Based on Learning. In *Design Automation Conference*, pages 420–426, June 1995.

[JNFSV97] Jawahar Jain, Amit Narayan, Masahiro Fujita, and Alberto L. Sangiovanni-Vincentelli. Formal verification of combinational circuits. In *International Conference on VLSI Design*, pages 218–225, January 1997.

[KAKS99] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. *IEEE Transactions on VLSI Systems*, 7(1):69–79, March 1999.

[KGP01] Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based Boolean Reasoning. In *Proceedings of the* 38th *Design Automation Conference*, pages 232–237, June 2001.

[KK97] Andreas Kuehlmann and Florian Krohm. Equivalence Checking Using Cuts and Heaps. In *Design Automation Conference*, pages 263–268, June 1997.

[KMS97] Henry Kautz, David McAllester, and Bart Selman. Exploiting Variable Dependency in Local Search. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, August 1997.

[KMSS00a] Joonyoung Kim, Jesse Whittemore João P. Marques-Silva, and Karem A. Sakallah. On Applying Incremental Satisfiability to Delay Fault Testing. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 380–384, March 2000.

[KMSS00b] Joonyoung Kim, Jesse Whittemore João P. Marques-Silva, and Karem A. Sakallah. On Solving Stack-Based Incremental Satisfiability Problems. In *Proceedings of International Conference on Computer Design*, pages 379–382, September 2000.

[KP94] Wolfgang Kunz and Dhiraj K. Pradhan. Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems - Test, Verification and Optimization. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 13(9):1143–1158, September 1994.

[KS97] Wolfgang Kunz and Dominik Stoffel. *Reasoning in Boolean Networks*. Kluwer Academic Publishers, June 1997.

[Kun93] Wolfgang Kunz. HANNIBAL: An efficient tool for logic verification based on recursive learning. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 538–543, November 1993.

[LA97] Chu Min Li and Anbulagan. Heuristics Based on Unit Propagation for Satisfiability Problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, August 1997.

[Lar92] Tracy Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, Jan 1992.

[LBMS01] Inês Lynce, Luís Baptista, and João P. Marques-Silva. Stochastic Systematic Search Algorithms for Satisfiability. In Henry Kautz and Bart Selman, editors, *The LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*, Electronic Notes in Discrete Mathematics. Elsevier Science Publishers, June 2001.

[Li00] Chu Min Li. Integrating Equivalency Reasoning into Davis-Putnam Procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 291–296. AAAI Press, July 2000.

[Lov78] D. W. Loveland. *Automated Theorem Proving: A Logical Basis.* North-Holland, 1978.

[Mat96] Yusuke Matsunaga. An Efficient Equivalence Checker for Combinational Circuits. In *Design Automation Conference*, pages 629–634, June 1996.

[McM92] Kenneth L. McMillan. *Symbolic model checking: An approach to the state explosion problem.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1992. CMU Technical Report CMU-CS-92-131.

[Mey75] Stuart L. Meyer. *Data Analysis For Scientists and Engineers.* Wiley and Sons, 1975.

[MJT$^+$99] Rajarshi Mukherjee, Jawahar Jain, Koichiro Takayama, Masahiro Fujita, Jacob Abraham, and Donald Fussell. An Efficient Filter–based Approach for Combinational Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18:1542–1557, November 1999.

[MMZ+01] Matthew H. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the* 38*th* *Design Automation Conference*, pages 530–535, June 2001.

[MS95] João P. Marques-Silva. *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits.* PhD thesis, University of Michigan, Ann Arbor, 1995.

[MS00] João P. Marques-Silva. Algebraic Simplification Techniques for Propositional Satisfiability. In Rina Dechter, editor, *Proceedings of the* 6*th* *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 1894 of *Lecture Notes in Computer Science*, pages 537–542. Springer, September 2000.

[MSeS99] João Marques-Silva and Luis Guerra e Silva. Algorithms for Satisfiability in Combinational Circuits Based on Backtrack Search and Recursive Learning. In *Workshop notes of The International Workshop on Logic Synthesis*, pages 227–241, June 1999.

[MSG99] João P. Marques-Silva and Thomas Glass. Combinational Equivalence Checking Using Satisfiability and Recursive Learning. In *Proceedings of the Design Automation and Test in Europe Conference*, pages 145–149, March 1999.

[MSS+91] Patrick McGeer, Alexander Saldanha, Paul R. Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Timing Analysis and Delay-Test Generation Using Path Recursive Functions. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pages 180–183, November 1991.

[MSS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[MSS00] João P. Marques-Silva and Karem A. Sakallah. Boolean Satisfiability in Electronic Design Automation. In *Proceedings of the* 37*th* *Design Automation Conference*, pages 675–680, June 2000.

[NASR01] Gi-Joon Nam, Fadi Aloul, Karem A. Sakallah, and Rob A. Rutenbar. A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints. In *Proceedings of the International Symposium on Physical Design*, April 2001.

[NSR99] Gi-Joon Nam, Karem A. Sakallah, and Rob A. Rutenbar. Satisfiability-Based Detailed FPGA Routing. In *Proceedings of the* 12th *International Conference on VLSI Design*, pages 574–577, January 1999.

[PB87] Paul W. Purdom and Cynthia A. Brown. Polynomial Average-Time Satisfiability Problems. *Information Sciences*, 41:23–42, 1987.

[PK00] Viresh Paruthi and Andreas Kuehlmann. Equivalence Checking combining a structural SAT-solver, BDDs and simulation. In *Proceedings of the IEEE International Conference on Computer Design*, pages 459–464, September 2000.

[Pre00] Steven D. Prestwich. A Hybrid Search Architecture Applied to Hard Random 3-SAT and Low-Autocorrelation Binary Sequences. In Rina Dechter, editor, *Proceedings of the* 6th *International Conference on Principles and Practice of Constraint Programming (CP 2000)*, volume 1894 of *Lecture Notes in Computer Science*, pages 337–352. Springer, September 2000.

[pro] Prover Technology. `http://www.prover.com`.

[Qui55] W. V. Quine. A Way to Simplify Truth Functions. *The American Mathematical Monthly*, 62(9):627–631, November 1955.

[Ric95] John A. Rice. *Mathematical Statistics and Data Analysis*. International Thompson Publishing, Second edition, 1995.

[Rob65] J. A. Robinson. A Machine Oriented Logic Based on the Resolution Principle. *Journal of the Association for Computing Machinery*, 12(1):23–41, January 1965.

[S$^+$92] Ellen M. Sentovich et al. SIS: A System for Sequential Circuit Synthesis. Technical Report UCB/ERL M92/41, ERL, College of Engineering, University of California, Berkeley, May 1992.

[SB01] Subarna Sinha and Robert K. Brayton. Robust and Efficient SPFD Computations. In *International Workshop on Logic and Synthesis (IWLS01)*, June 2001.

[SBSV96] Paul Stephan, Robert K. Brayton, and Alberto L. Sangiovanni-Vincentelli. Combinational Test Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(9):1167–1176, September 1996.

[SC01] Laurent Simon and Philippe Chatalic. SatEx: A Web-based Framework for SAT Experimentation. In Henry Kautz and Bart Selman, editors, *The LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*, volume 9 of *Electronic Notes in Discrete Mathematics*. Elsevier Science Publishers, June 2001. `http://www.lri.fr/~simon/satex/satex.php3`.

[Sht00] Ofer Shtrichman. Tuning SAT Checkers for Bounded Model Checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the* 12th *International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2000.

[Sht01] Ofer Shtrichman. Pruning Techniques for the SAT-based Bounded Model Checking Problem. In *Proceedings of the* 11th *Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME 2001)*, September 2001. *To appear.*

[SKC96] Bart Selman, Henry A. Kautz, and Bram Cohen. Local Search Strategies for Satisfiabilty Testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996. *D.S. Johnson and M. A. Trick Eds.*

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446. AAAI Press, July 1992.

[SS90] Gunnar Stålmarck and M. Säflund. Modeling and verifying systems and software in propositional logic. In B.K. Daniels, editor, *Proceedings of Safety of Computer Control Systems (SAFECOMP'90)*, pages 31–36. Pergamon Press, October 1990.

[SS00] Mary Sheeran and Gunnar Stålmarck. A tutorial on Stålmarck's proof procedure for propositional logic. *Formal Methods in System Design*, 16(1):23–58, January 2000.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stålmarck. Checking safety properties using induction and a SAT-solver. In Warren A. Hunt and Steven D. Johnson, editors, *Proceedings of the 3^{rd} International Conference on Formal Methods in Computer Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 108–125. Springer, November 2000.

[Stå] Gunnar Stålmarck. A system for determining propositional logic theorems by applying values and rules to triplets that are generated from a formula, 1989. Swedish patent no. 467 076(1992), U.S. patent no. 5 276 897(1994), European patent no. 0404 454(1995).

[STS88] Michael H. Schulz, Erwin Trischler, and Thomas M. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 7(1):126 –137, January 1988.

[SV93] Thomas Schiex and Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, pages 48–55, 1993.

[TG99] Paul Tafertshofer and Andreas Ganz. SAT Based ATPG Using Fast Justification and Propagation in the Implication Graph. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 139–146, November 1999.

[TGH97] Paul Tafertshofer, Andreas Ganz, and Manfred Henftling. A SAT-Based Implication Engine for Efficient ATPG, Equivalence Checking and Optimization of Netlists. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 648–657, November 1997.

[Tse68] G. S. Tseitin. On the Complexity of Derivation in Propositional Calculus. In A. O. Silenko, editor, *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125. Consultants Bureau, New-York-London, 1968. Part II.

[VB99] Miroslav N. Velev and Randal E. Bryant. Superscalar Processor Verification Using Efficient Reductions of the Logic of Equality with Uninterpreted Functions to Propositional Logic. In L. Pierre and T. Kropf, editors, *Proceedings of Correct Hardware Design and Verification Methods (CHARME '99)*, volume 1703 of *Lecture Notes in Computer Science*, pages 37–53. Springer, September 1999.

[VB00] Miroslav N. Velev and Randal E. Bryant. Formal Verification of Superscalar Microprocessors with Multicycle Functional Units, Exceptions, and Branch Prediction. In *Proceedings of the* 37th *Design Automation Conference*, pages 112–117, June 2000.

[vE97] C. A. J. van Eijk. *Formal Methods for the Verification of Digital Circuits.* PhD thesis, Eindhoven University of Technology, Dept. of Electrical Engineering, Eindhoven, Netherlands, 1997.

[Vel00] Miroslav N. Velev. Formal Verification of VLIW Microprocessors with Speculative Execution. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the* 12th *International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 296–311. Springer, 2000.

[WAH01] Poul Frederick Williams, Henrik Reif Andersen, and Henrik Hulgaard. Satisfiability Checking Using Boolean Expression Diagrams. In Tiziana Margaria and Wang Yi, editors, *Proceedings of the* 7th *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 39–51. Springer, April 2001.

[WBCG00] Poul F. Williams, Armin Biere, Edmund M. Clarke, and Anubhav Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In E. Allen Emerson and A. Prasad Sistla, editors, *Proceedings of the* 12th *International Conference on Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2000.

[WKS01] Jesse P. Whittemore, Joonyoung Kim, and Karem A. Sakallah. SATIRE: A New Incremental Satisfiability Engine. In *Proceedings of the* 38th *Design Automation Conference*, pages 542–545, June 2001.

[WP79] Thomas W. Williams and Kenneth Parker. Testing Logic Networks and Designing for Testability. *Computer*, pages 9–21, October 1979.

[WR98] R. Glen Wood and Rob A. Rutenbar. FPGA Routing and Routability Estimation via Boolean Satisfiability. *IEEE Transactions on VLSI Systems*, 6(2):222–231, June 1998.

[Yan91] Saeyang Yang. Logic Synthesis and Optimization Benchmarks User Guide. Technical report, Microelectronics Center of North Carolina, January 1991. Version 3.0.

[Zha97] Hantao Zhang. SATO: An Efficient Propositional Prover. In William McCune, editor, *Proceedings of the International Conference on Automated Deduction (CADE'97)*, volume 1249 of *Lecture Notes in Artificial Intelligence*, pages 272–275. Springer, July 1997.

[ZMAM99] Peixin Zhong, Margaret Martonosi, Pranav Ashar, and Sharad Malik. Using Configurable Computing to Accelerate Boolean Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):861–868, June 1999.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in a Boolean Satisfiability Solver. In *Proceedings of the IEEE/ACM International. Conference on Computer Aided Design*, November 2001. *To appear.*

[ZRP97] Jian-Kun Zhao, Elizabeth M. Rudnick, and Janak H. Patel. Static Logic Implication with Application to Redundancy Identification. In *Proceedings of the* 15th *IEEE VLSI Test Symposium*, pages 288–293, April 1997.

[ZS96] Hantao Zhang and Mark E. Stickel. An Efficient Algorithm for Unit Propagation. In *Proceedings of the Fourth International Symposium on Artificial*

Intelligence and Mathematics (AI-MATH'96), Fort Lauderdale (Florida USA), 1996.

Index

assignment, 9
  complete, 9
  partial, 9
ATPG problem, 39
ATPG-SAT, 40
average-time analysis, 44

backtracking, 18
  chronological, 18
  non-chronological, 5, 18, 100
  simple, 46
BCP, 12
BDD, 71, 78
Boolean constraint propagation, 12, 28
Boolean network, 22
Boolean variable, 8
bounded model checking, 33
branch and bound, 11

candidate pair, 84
Chaff, 28
circ, 65
circuit width
  forward width, 71
  reverse width, 71
CIRCUIT-SAT, 23, 39
circuit-width, 71
clause, 8
CNF formula, 9
combinational
  ATPG, 36
  equivalence checking, 74, 76
  logic circuit, 22
  verification, 76
complexity analysis, 69
complexity index, 41
conflict, 28
  analysis, 16, 93
  based learning, 5, 18
  clause, 12
  clause recording, 18, 48, 102
  condition, 12
conflict driven learning, 28
conjunctive normal form, 9
consensus, 14
consistency function, 23, 42
curve-fitting, 68
  least squares error, 68
  normalization error, 69
  truncation error, 69
cut-width, 6, 52
  minimum, 52
cutpoint, 77

Davis-Putnam algorithm, 15

decision
  assignment, 12
  level, 12
  variable, 12
deduced assignment, 12
DPLL algorithm, 12, 91

empty clause, 10, 15
equivalence, 82
  class, 83
  cut, 77, 82
  modulo complementation, 82
essential point, 7, 97

failure-driven assertion, 97
false negative, 77
fanin, 22
  transitive, 22
fanout, 22
  transitive, 22
fault
  boundary, 39
  collapsing, 62
  dominance, 62
  equivalence, 62
  single stuck-at, 38
faulted circuit, 39
free literal, 10

gen, 65
global hypothesis, 84
  class, 84
GRASP, 16, 28, 86
ground resolution, 15
GSAT, 32, 114

HANNIBAL, 33, 78, 87
HeerHugo, 29
HMETIS, 62
horn formula, 41
hyper-edge, 51
hypergraph, 51

implication, 12
implication graph
  data-structure, 29, 33, 87
incremental satisfiability, 28
induction, 34
injured clause, 52

k-bounded circuit, 43, 60
Ketchum, 34

left branch assignment, 12
literal, 8
log-bounded width circuit, 6, 59
logic circuit, 22
logic value, 8

microprocessor verification, 34
min-cut linear arrangement, 62
minterm, 93
miter, 77

NEMESIS, 32
net, 38
non-chronological backtracking, 28
NP-complete, 1, 36, 62

PODEM, 29

primary input, 22
primary output, 22
problem symmetry, 7, 91
  example, 95
product-of-sums, 24
propositional
  formula, 20
  variable, 8
propositional satisfiability, 1
PROVER, 34
pure literal rule, 14, 99
pure polarity, 99

q-Horn formula, 41

random vector simulation, 62, 84, 114
recursive learning, 5, 29, 74, 78, 87
relevance based learning, 28
RELSAT, 16, 28
replacability relationship, 76, 78
resolution, 15, 27, 101
resolvent, 101
right branch assignment, 12

SAT, 1
  applications, 2
  applications in EDA, 3
  problem, 10
Sat-Ex, 35
SATIRE, 28
satisfiability problem, 10
satisfiable, 10
satisfied, 10
satisfying assignment, 10, 23
SATLIB, 35
SATLIVE, 35
SATO, 28
SATZ, 30
search tree, 12
simple gate, 24
SIS, 63, 88
SIVA, 34
SOCRATES, 29
stuck-at fault, 38
Stålmarck's method, 28, 34
sub-formula, 48
  consistent, 48
  distinct consistent, 51
supercubing
  operator, 103
  rule, 92, 104
symmetric point, 97
synthetic benchmark generation, 65

tautology, 15
TEGUS, 32
TIP, 32
transitive fanin, 22
transitive fanout, 22
tree, 60
tree-width, 113
Tseitin's transformation, 21
  for logic circuits, 24

unit clause, 10
unit literal rule, 12
unresolved, 10
  clause, 99

unsatisfiability cube, 94
unsatisfiable, 10
unsatisfied, 10

worst case complexity analysis, 4
WSAT, 32, 114