UM MODELO ESTATÍSTICO MULTIVARIADO PARA PREVER O COMPORTAMENTO DE HEURÍSTICAS EM VERIFICAÇÃO FORMAL

GEÓRGIA PENIDO SAFE

UM MODELO ESTATÍSTICO MULTIVARIADO PARA PREVER O COMPORTAMENTO DE HEURÍSTICAS EM VERIFICAÇÃO FORMAL

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

Orientador: Claudionor Nunes Coelho Jr

Belo Horizonte

Dezembro de 2011

GEÓRGIA PENIDO SAFE

A MULTIVARIATE CALIBRATION MODEL TO PREDICT FASTER FORMAL VERIFICATION HEURISTICS

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Doctor in Computer Science.

Advisor: Claudionor Nunes Coelho Jr

Belo Horizonte

December 2011

© 2011, Geórgia Penido Safe.
Todos os direitos reservados.

Safe, Geórgia Penido
S128m A multivariate calibration model to predict faster formal verification heuristics / Geórgia Penido Safe. — Belo Horizonte, 2011
xxiv, 108 f. : il. ; 29cm

Tese (doutorado) — Federal University of Minas Gerais
Orientador: Claudionor Nunes Coelho Jr

1. Computaão - Teses. 2. Programaão paralela - Teses. 3. Verificaão funcional - Teses. I. Titulo.

CDU 519.6*31(043)

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

FOLHA DE APROVAÇÃO

Um modelo estatístico multivariado para prever o comportamento de heurísticas em verificação formal

GEORGIA PENIDO SAFE

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

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

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

PROF. IVAN SARAIVA SILVA
Departamento de Informática e Estatística - UFPI

PROF. RICARDO AUGUSTO DA LUZ REIS
Instituto de Informática - UFRGS

PROF. SÉRGIO VALE AGUIAR CAMPOS
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 02 de dezembro de 2011.

To my parents, Jose Luciano and Glete,

For teaching me what really matters

To my husband and my daughter, Gustavo and Lavinia,

For being the reason of everything

Acknowledgments

First of all, I would like to thank God for all He has given me. Without Him, none of this would have been possible.

Thanks to my supervisor, who was always beside me giving all his support and friendship. I will never forget.

Thanks to all my family: my husband and Lavinia for the comprehension and support, my parents for the sincere opinions, my sister Erika for trying to help, my sister Andresa and my brother Luciano, for the good wishes.

Thanks to Professor Luis Filipe, for helping with all his experience in the paper elaboration.

It is a pleasure for me to work with all the wonderful people from Jasper. Thanks to all of you for every smile I received when my paper was accepted and for all support. Thanks to Celina, for helping in the data collection.

Thanks to Jose Nacif, for the discussions, support and useful hints.

Thanks to Renata, for the patience and guiding in all university processes. People like you make our department one of the best in our country.

Thanks to all professors that I consider my friends: José Marcos, Antônio Otávio and Antônio Alfredo, for being part of all this.

Thanks for the examinators, for the valuable ideas and corrections.

Resumo

Verificação funcional é "o" principal gargalo na produtividade de empresas desenvolvedoras de chips. Como a verificação funcional é um problema NP-completo, ela depende de um grande número de heurísticas e seus parâmetros de configuração ("resolvedores"). Normalmente o número de resolvedores disponíveis excede em muito o poder de processamento disponível. Com o advento da programação paralela, a verificação funcional pode ser otimizada através da seleção dos melhores n resolvedores para rodar em paralelo, aumentando assim a chance de se alcançar o término da verificação. Este trabalho apresenta um modelo estatístico baseado em métricas estruturais para construir estimadores de tempo de verificação para os resolvedores, permitindo então a seleção dos n melhores resolvedores para rodar em paralelo. Esta metodologia considera tanto o tempo de execução estimado dos resolvedores quanto a correlação entre estes resolvedores. Resultados confirmaram que a metodologia pode ser um mecanismo muito rápido e eficaz para a seleção dos melhores resolvedores, aumentando a chance de se resolver o problema.

Abstract

Functional verification is “the” major design-phase bottleneck for silicon productivity. Since functional verification is an NP-complete problem, it relies on a large number of heuristics with associated parameters (engines). With the advent of parallel processing, formal verification can be optimized by selecting the best n engines to run in parallel, increasing the chance of reaching verification successful termination. In this work, we present an statistical model to build engine estimators based on structural metrics and to select n engines to run in parallel. The methodology considers both engines' estimated performance and engines' correlation. Results confirmed that the methodology can be a very quick selection mechanism for parallelization of engines in order to increase the chance of running the best engines to solve the problem.

List of Figures

1.1 Transistor counts for integrated circuits plotted against their dates of introduction. . . . 2
1.2 The verification gap leaves design potential unrealized. . . . 3
1.3 Re-spin frequency in North America, as brought by Fujita Lab. . . . 3
1.4 Re-spin causes in North America, as brought by Fujita Lab. . . . 4

2.1 Formal verification classification overview. . . . 14
2.2 Symbolic model checking techniques overview. . . . 16
2.3 Decision tree diagram. . . . 17
2.4 Reductions: merge duplicates and eliminate redundancy. . . . 18
2.5 Final BDD. . . . 18
2.6 Graphical visualization of k-induction algorithm. . . . 25
2.7 Graphical visualization of CEX-based abstraction algorithm. . . . 29
2.8 Graphical visualization of Proof-based abstraction algorithm. . . . 31
2.9 Graphical visualization of interpolation algorithm. . . . 33

3.1 Globally linear nature of linear regression. . . . 40
3.2 1-NN and 5-NN neighborhood. . . . 42
3.3 Locally constant nature of kNN regression. . . . 44
3.4 Basic principle of PCA in two dimensional case. . . . 45

4.1 The investor‘s opportunity set. . . . 48
4.2 Indifference curves. . . . 49
4.3 Investor equilibrium. . . . 51
4.4 Investor’s opportunity set with several alternatives. . . . 52
4.5 Risk and return possibilities for various assets and portfolios. . . . 59
4.6 The efficient frontier. . . . 60
4.7 Combinations of the riskless asset in a risky portfolio. . . . 60

5.1 Methodology to identify variables with polynomial effect over an engine. . 74

6.1 Polinomial effect for smaller values versus exponential effect for larger values. 84
6.2 Results comparison for selection heuristics. 88

A.1 SCOAP controllability equations. 103
A.2 SCOAP observability equations. 104
A.3 Flip-flop. 104
A.4 SCOAP observability equations. 105

List of Tables

3.1 Main differences between linear and k-nearest neighbors methods. 44

4.1 Frequency function. 53

5.1 Symbols convention. 68
5.2 r-squared for engine estimators with all data. 76

6.1 Percentage of times smaller distance matched correctly for each group at the end of validation process. 85
6.2 Percentage of time of each kind of match for selection based in execution time. 86
6.3 Percentage of time of each kind of match for selection based in execution time and variance. 87
6.4 Mean r-squared for engine estimators after cross-validation. 88

Contents

Acknowledgments **xi**

Resumo **xiii**

Abstract **xv**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**

1.1 Motivation and Goal 1

1.2 Problem Definition 5

1.3 Related work 8

1.4 Summary of the thesis 9

1.4.1 Chapter 2 9

1.4.2 Chapter 3 10

1.4.3 Chapter 4 10

1.4.4 Chapter 5 10

1.4.5 Chapter 6 11

1.4.6 Chapter 7 11

1.4.7 Appendix 1 11

1.5 Thesis contributions 11

2 Formal Verification **13**

2.1 Theorem proving 13

2.2 Formal model checking 14

2.2.1 Formal equivalence checking 14

2.2.2 Formal property checking 15

2.3 Symbolic Model Checking . . . 15
2.3.1 Infrastructure Layer . . . 16
2.3.2 Falsification and Proof Methods Layer . . . 23
2.3.3 Abstraction Layer . . . 27
2.3.4 Proof engines . . . 33

3 Statistical Learning 35
3.1 Multivariate regression . . . 36
3.2 Linear regression . . . 37
3.3 Nearest neighbor method . . . 40
3.4 Principal component analysis . . . 43

4 Mean Variance Portfolio Theory 47
4.1 The Opportunity Set . . . 47
4.2 The Indifference Curves . . . 49
4.3 Opportunity Set Under Risk . . . 51
4.3.1 Return Distribution . . . 52
4.3.2 Variance . . . 53
4.4 Combination of Assets . . . 54
4.5 Efficient Frontier . . . 58

5 A Multivariate Calibration Model to Predict Faster Verification Heuristics 67
5.1 Independent Variables ($\vec{x}$) . . . 68
5.2 Data set . . . 71
5.2.1 Data set partition . . . 71
5.3 Engine Estimators . . . 72
5.3.1 Linearization of Exponential Equations . . . 72
5.3.2 Polynomial Effect of Variables . . . 73
5.3.3 Multivariate regression . . . 75
5.4 Engines' selection mechanisms . . . 77
5.4.1 Maximizing Performance . . . 78
5.4.2 Minimizing Correlation . . . 78
5.4.3 Maximizing Performance and Chance of Success . . . 79

6 Results 83
6.1 Multivariate regression . . . 83
6.2 Smallest distances . . . 84

6.3 Selection based on engine's time 85
6.4 Selection based on engine's time and correlation 86
6.5 Summarization 87

7 Conclusions 89

Bibliography 93

A Testability measures 101
A.1 SCOAP (Sandia Controllability Observability Analysis) [42] 102
A.2 COP (Controllability/Observability Program) [18] 105
A.3 PREDICT [72] 105
A.4 VICTOR [69] 106
A.5 STAFAN (Statistical Fault Analysis) [52] 106
A.6 TMEAS (Testability Measure Program) [43] 106
A.7 CAMELOT (Computer-Aided Measure for Logic Testability) [13] . . . 108

Chapter 1

Introduction

1.1 Motivation and Goal

Electronic designs have been growing rapidly in both device count and functionality, concretizing Moore's Law, as it can be seen in figure 1.1.

This growth has been enabled by deep sub-micron fabrication technology, and fueled by expanding consumer electronics, communications and computing markets. A major impact on the profitability of electronic designs is the increasing productivity gap. That is, what can be designed is lagging behind what silicon is capable of delivering [78], as shown in figure 1.2 [11].

Functional verification is already "the" major design-phase bottleneck, and it will only get worse with the bigger capacities on the horizon [50].

Broadly speaking, the verification crises can be attributed to the following interacting situations [78]:

- Verification complexity grows super-linearly with design size,
- The increased use of software, which has intrinsically higher verification complexity,
- Shortened time-to-market,
- Higher cost of failure (low profit margin).

A consequence of this verification crisis is that re-spins are becoming more frequent, as it can be seen in figure 1.3, and the main causes are logic and functional errors, as shown by statistics in figure 1.4. These errors could be reduced by a better functional verification.

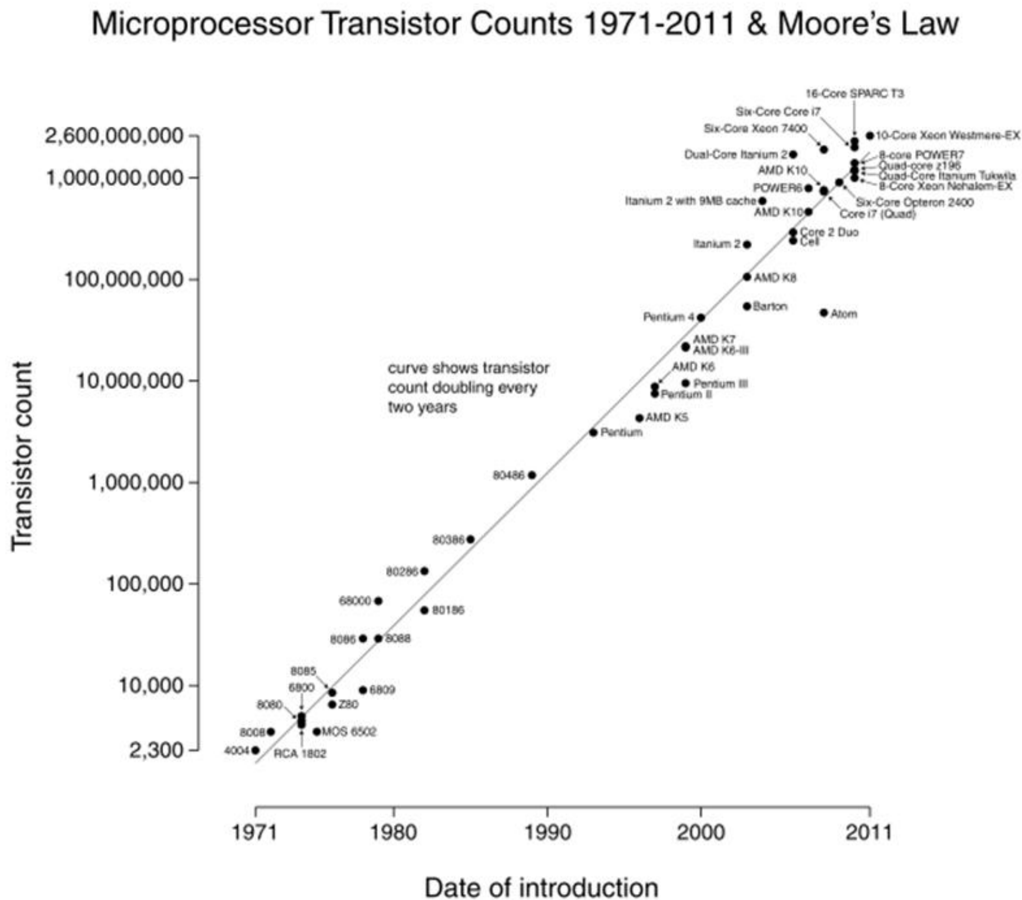

Figure 1.1. Transistor counts for integrated circuits plotted against their dates of introduction.

Automated formal verification arrived to aid. Allowing the exhaustive exploration of all possible behaviors, automated formal verification is easy to use and increases the reliability of the design.

Despite all the recent advances in formal verification technology [33], formal verification is an NP-complete problem in the size of the trace description (Binary Decision Diagrams state-explosion problem) and in the verification time (SAT solvers). Therefore, as many other NP-complete problems, practical solutions implement heuristics to try to solve efficiently such problems.

In this work, each pair "heuristic" $\times$ "configuration parameters" will be referred to as an engine. Since in functional verification, complexity is often measured by the size of the design, which is exponential in the number of storage elements (flip-flops) in the design [78], the performance of an engine is completely dependent on the design size and on the property complexity to be verified. It becomes then necessary to identify and apply the best engines to each property to be proved. The application of the

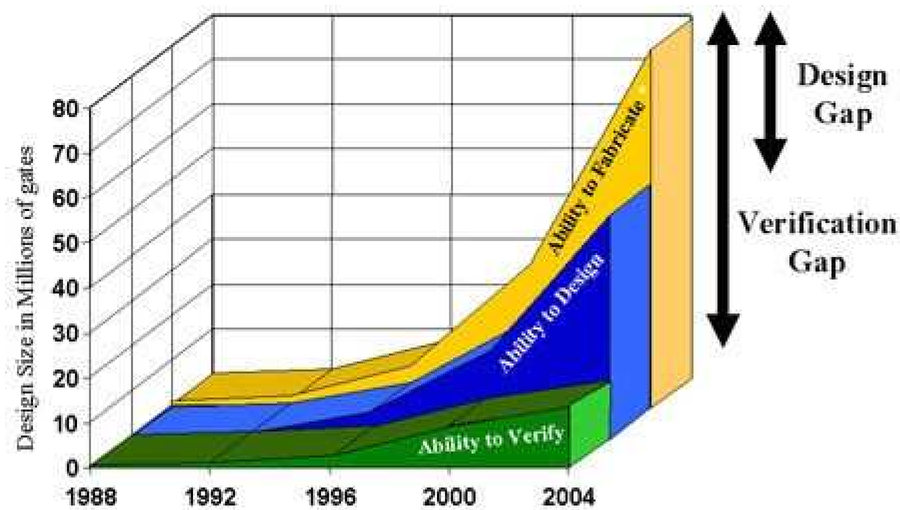

Figure 1.2. The verification gap leaves design potential unrealized.

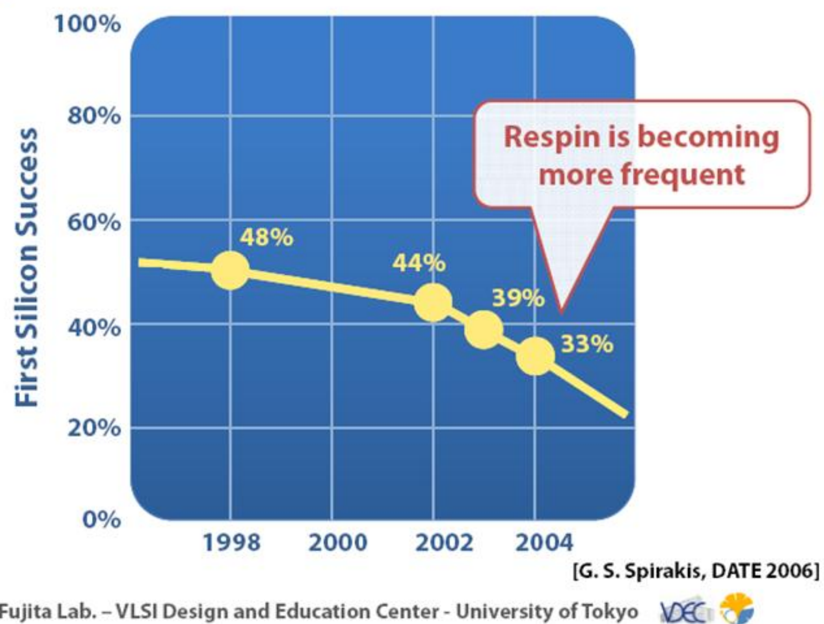

Figure 1.3. Re-spin frequency in North America, as brought by Fujita Lab.

best engines will speed up the verification process and increase the chance of successful termination. However, selecting the best heuristics and parameters to prove a property is a complex problem.

Approaches used today to solve this problem are based mainly on brute force or on verification engineers' feelings. By brute force, a user starts one proof process/thread

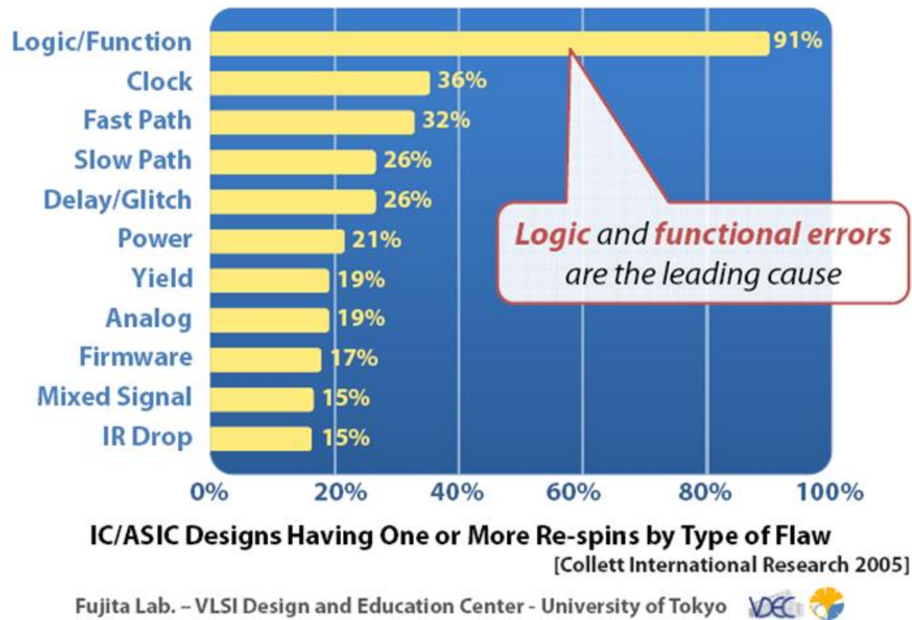

Figure 1.4. Re-spin causes in North America, as brought by Fujita Lab.

for each available engine and gets the first result available. However, since engines and associated parameters may be much more numerous than the processing power available, this approach can easily be shown not to scale. On the other hand, verification engineers may have a feeling of the best heuristic to prove a property, but with the increase in design size, trusting in these feelings becomes a big risk.

With the advent of parallel processing (simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads), the possibility of running multiple processes in parallel became more transparent and viable. As a consequence, relying on the use of multiple processes/threads in order to get the best results arose as a way to improve formal verification.

In general, there are a large pool of engines, but only a small number of processing cores. The problem then becomes selecting the best subset of engines that will increase the chance of solving the problem, which in this context means reaching a proof conclusion.

The goal of this work is to generate a forecasting mechanism to classify and select the n best engines to run in parallel, in order to maximize the chance of proof conclusion or successful termination.

The main motivation is the possibility of speeding up verification of digital cir-

cuits, given the verification crisis and the existence of so many different engines to implement formal verification.

In a verification tool that implements several formal verification heuristics with associated parameters, selecting the best heuristics and parameters to prove a property becomes a complex problem.

We know that designs are getting bigger and bigger and that time to verify systems is crucial to improve their time-to-market. Therefore, choosing engines with greater chance of solving the problem turns out to be an important question.

1.2 Problem Definition

A Hardware Description Language (HDL) is a standard text-based expression of the temporal behavior and/or (spatial) circuit structure of an electronic system. The vast majority of modern digital circuit-design revolves around an HDL-description of the desired circuit, device, or subsystem, with VHDL (VHSIC Hardware Description Language) and Verilog HDL being the two dominant HDLs. SystemVerilog is a combined HDL and Hardware Verification Language (HVL) based on extensions to Verilog.

A simple example of two flip-flops swapping values every clock in Verilog is:

```
module toplevel(clock,reset);
 input clock;
 input reset;

 reg flop1;
 reg flop2;

 always @ (posedge reset or posedge clock)
 if (reset)
   begin
     flop1 <= 0;
     flop2 <= 1;
   end
 else
   begin
     flop1 <= flop2;
     flop2 <= flop1;
   end
```

```
endmodule
```

Property checking takes a design and a property which is a partial specification of the design and its surroundings, and proves or disproves that the design satisfies the property. A property is essentially an abstracted description of the design, and it acts to confirm the design through redundancy.

Properties may be specified in many ways. A Hardware Verification Language, or HVL, is a programming language used to verify the designs of electronic circuits written in a hardware description language. OpenVera [6], Specman(e) [3], and SystemC [5] are the most commonly used HVLs. SystemVerilog [1] attempts to combine HDL and HVL constructs into a single standard. Open Verification Library (OVL) [2] is a library of property checkers for digital circuit descriptions written in popular HDLs. Property Specification Language (PSL) [4] is a language developed by Accellera for specifying properties or assertions about hardware designs.

SystemVerilog, for example, supports assertions, assumptions and coverage of properties. An assertion specifies a property that must be proven true. An assumption establishes a condition that a formal logic proving tool must assume to be true. In simulation, both assertions and assumptions are verified against test stimulus. Property coverage allows the verification engineer to verify that assertions are accurately monitoring the design.

A SystemVerilog assertion example is:

```
property req_gnt;
@(posedge clk)
req |=> gnt;
endproperty

assert_req_gnt: assert property (req_gnt)
                else $error("req not followed by gnt.");
```

This example shows an implication operator $|=>$. The clause to the left of the implication is called the antecedent and the clause to the right is called the consequent. Evaluation of an implication starts through repeated attempts to evaluate the antecedent. When the antecedent succeeds, the consequent is attempted, and the success of the assertion depends on the success of the consequent. In this example, the consequent won't be attempted until *req* goes high, after which the property will fail if *gnt* is not high on the following clock.

A program that checks a property is also called a model checker, referring the design as a computational model of a real circuit.

The idea behind property checking is to search the entire state space for points that fail the property. If such a point is found, the property fails and the point is a counterexample. Next, a waveform derived from the counterexample is generated, and the user debugs the failure. Otherwise, the property is satisfied.

What makes property checking a success in industry are efficient SAT solvers or symbolic traversal algorithms that enumerate the state space implicitly (see chapter 3).

Each group of heuristics and algorithms (with its set of configuration parameters) used to accomplish one proof can generate a different engine e (or solver).

Some parameter examples from the SAT-solving world include decision variable and phase selection, clause deletion and initial variable ordering. The complex effects and iterations between these parameters render the design and implementation of a high-performance decision procedure (or, indeed, a high-performance heuristic algorithm for any NP-hard problem) so challenging that in many ways the process resembles an art rather than a science [51].

The behavior and performance of each engine is highly dependent on circuit design, or the property to be proved, whose behavior can be characterized by a set of independent variables $\vec{x}$ and a set of unknown variables $\vec{U}$. We consider $\vec{U}$ to be insignificant in the problem context. However, since the problem is hard, we will always have a chance that the chosen independent variables won't be enough to explain the behavior of a given verification problem instance.

Some independent variables examples are the number of flip-flop bits and the number of counter bits in the design.

Given

- a design and a property to be proved expressed by vector $\vec{x}$ of selected independent variables that are supposed to explain the behavior of the verification problem,
- N proof heuristics $E_1, E_2, ...E_N$, each one configurable by vector $\vec{p_i}$ of parameters, naming engines $e_1, e_2, ...e_N$,

the objective of this work is to statistically compute engines estimators

- $\tilde{e}_i(\vec{x}) \propto e_i$

to select n engines in such a way that

1. $n < N$

2. $e_1, ..., e_n$ is the n set of engines with the biggest chance of reaching to a proof conclusion.

where function $\tilde{e}_i(\vec{x})$ returns the estimated time of engine i to prove a property whose design is characterized by independent variables $\vec{x}$. In fact, the estimated execution time of engine i is a function of $\vec{x}$ and $\vec{U}$, where $\vec{U}$ refers to some unknown variables. As already stated, we approximate $\tilde{e}(\vec{x}, \vec{U})$ as $\tilde{e}(\vec{x})$. This approximation is valid if the error introduced by dropping $\vec{U}$ is small.

1.3 Related work

Traditionally, formal verification has employed engines based on SAT and BDD algorithms. Such algorithms have been constructed with several heuristics, captured by parameters (MiniSAT [28], Cudd [76], GRASP [55], zchaff [81], SATO [79], Berk-Min [30]). However, these engines are usually executed in a monolithic way, i.e. with no or with few parameters, with exception of the work [51], which tunes the heuristics parameters. Besides that, there are many algorithms employed in formal verification (such as reachability analysis, induction, bounded model checking and proof-based abstraction), each of them with its limitations and advantages, depending on the applications.

McMillan et al. works [8; 9; 60; 58] compare many model checking techniques on a set of benchmark model checking problems derived from microprocessor verification. As it can be seen, performance of each technique depends greatly on circuit design, which generates different Conjunctive Normal Form (CNF) representations that may be best solved by different approaches.

Automatic parameterization of formal verification engines has only attracted attention recently. The closest work is [51], which presents a methodology to tune heuristic parameters, improving a specific heuristic performance. They employed ParamILS, a parameter optimization tool, to enhance SPEAR, a high-performance modular arithmetic decision procedure and SAT solver. Results show speedups between 4.5 and 500 in comparison to a manually-tuned version of SPEAR that already reached the performance of a state-of-the-art SAT solver.

In contrast, this work presents a migration from monolithic code (heuristic with no parameters) to modular parameterized code, by allowing the selection of optimal heuristics among many different heuristics and parameters configurations. The big

enhancement here is that we deal well with situations where a heuristic may perform well to a design, but very badly to another design (in this case, even with parameter optimization, speed ups can be difficult to reach). Being able to point to an optimal heuristic and parameter configuration among all available ones is a great step in the verification process.

In the area of semiconductor manufacturing, increased yield and improved product quality result from reducing the amount of wafers produced under suboptimal operating conditions. Some works employing multivariate statistical process control (MSPC) in the semiconductor manufacturing process in order to improve fault detection have been found. As examples, [22] presents a complete MSPC application method that combines recent contributions to the field, including multiway principal component analysis (PCA), recursive PCA, fault detection using a combined index, and fault contributions from Hotelling's T^2 statistic. [67] presents an enhancement to previous work that consisted of a fault detection method using the k-nearest-neighbor rule (FD-kNN). To reduce memory requirements and computation time of the proposed FD-kNN method while still keeping its advantage of handling nonlinear and multimode data, an improved FD-kNN algorithm based on principal component analysis, denoted as PC-kNN has been proposed.

1.4 Summary of the thesis

1.4.1 Chapter 2

Chapter 2 presents a global review on Formal Verification and a detailed review on Symbolic Model Checking, pointing the most used algorithms and heuristics.

The main goal is to explain some of the main model checking heuristics, emphasizing the fact that one heuristic can have many variations, controlled by their configuration parameters. The parameters direct the heuristics in the decisions to take.These decisions bring up different behaviors, which can improve performance for some designs, but not for others. The existence of a large number of engines due to the large number of heuristics and their variations (parameters) and the fact that heuristics can be good for one design, but bad for another, is one of the justifications to apply the proposed methodology: choose the engines with the biggest chance of success to prove a property.

The engines used in this work to validate the proposed methodology have been selected from a pool of model checking engines.

1.4.2 Chapter 3

Chapter 3 presents a review on Statistical Learning techniques, detailing methods that can be used to build prediction models in a multivariate scenario.

In order to choose the n engines with the best chance of success, it is mandatory to have a time prediction model for each engine, since the main aspect of the engine performance is its execution time. We can consider that the most efficient engines have smaller execution times (independently of the proof result: proven or falsified), and good engine execution time estimators will influence in the quality of this work.

A property to be proven will be expressed by a set of independent variables, that are comprised by metrics gotten from the design. Therefore, a multivariate linear regression model will be used to generate the engine estimators.

The main goal is to explain the statistical theory upon which the engine estimators have been generated.

1.4.3 Chapter 4

Chapter 4, Mean Variance Portfolio Theory, presents an interesting finance theory which will be applied in our solution.

Choosing engines by their performance (execution times) seems to be a good methodology, but the correlation between engines is also a factor that needs to be taken into account. Correlated engines tend to have similar behaviors and all selected correlated engines could lead to the same final result: if they all have longer execution time, we could arrive to no proof conclusion. In order to maximize the chance of success, the proposed solution will try to maximize performance and minimize correlation between selected engines.

This chapter explains the Mean Variance Portfolio Theory and their formulas that will be applied in our solution.

1.4.4 Chapter 5

Chapter 5 presents designed solution to solve problem defined.

The main goal is to define the methodology of the solution, applying the presented theories. The solution comprises the estimators generation, taking into account the polynomial effect of independent variables, the linearization of exponential equations and the model to select engines based on estimators' performance and correlation.

1.4.5 Chapter 6

Chapter 6 presents experimental results.

Two solutions were tested: (1) maximize performance and (2) maximize performance and minimize correlation. Both methodologies proved to be efficient, with speed ups of more than 3 times compared to a not optimal selection of engines.

1.4.6 Chapter 7

Chapter 7 concludes the work, presenting possible future works.

1.4.7 Appendix 1

Appendix 1 presents some testability measures that can be used as input data to the predictor.

1.5 Thesis contributions

This work's contributions are:

- Statistical model to predict faster proof heuristics

  A model will be proposed to predict faster proof engines (heuristics with associated parameters), given a design and a property to be proved. Multivariate statistics techniques will be used to generate this prediction model.

- Analysis of variable effect

  Although variables seem to have an exponential effect over the prediction model due to inheritance of NP-completeness of the verification process, heuristics can be affected in different ways by the metrics collected from the design and the property to be proven. It is possible, for example, to have a metric that has an exponential effect on one heuristic, while having a polynomial effect on another heuristic. This work tries to identify polynomial effects of variables through the analysis of the prediction model error.

- Analysis of engine correlation

  Analysis of engines correlation in order to guide selection of the n engines with the best chance of solving the problem.

Chapter 2

Formal Verification

Verification importance has emerged as an indispensable phase of digital hardware design development process. The increasingly competitive market makes cost of chip failure enormous, forcing traditional "black-box" verification methodology to give place to a "white-box" methodology.

In order to accomplish verification of designs that get more and more complex, traditional simulation faces the drawback that every time more simulation cases are necessary to arrive to accepted coverage levels. In addition, the increase of simulation sequences, the test-benches, makes verification process longer and more difficult, considering the effort to debug the test-benches, what may require the understanding of design characteristics, the features being tested and the simulation results.

Formal verification comes to solve this gap between design complexity and verification efforts.

Formal verification ensures that a design is correct with respect to a property that can be proved. The correctness of a property is based on its specification. Mathematical methods are used to prove or disprove a property.

Formal verification can be broadly classified as shown in Figure 2.1. This chapter gives a high level view of formal verification and a detailed view of Symbolic Model Checking.

2.1 Theorem proving

One of the earliest approaches to formal verification was to describe both the implementation and the specification in a formal logic. The correctness was then obtained by, in the logic, proving that the specification and the implementation were suitably related [17].

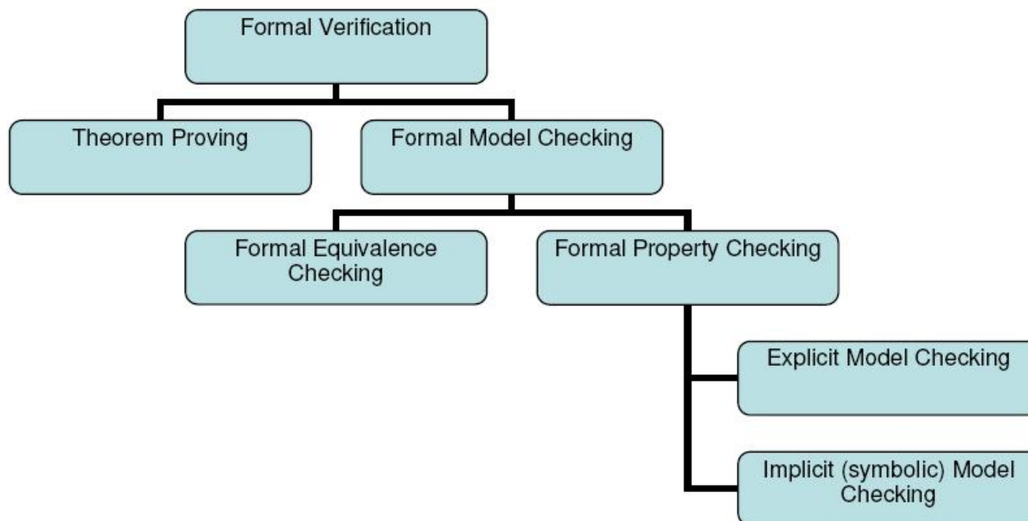

Figure 2.1. Formal verification classification overview.

Unfortunately, theorem proving technique requires a large amount of effort by the user developing specifications of each component and in guiding the theorem proving user through all the lemmas.

2.2 Formal model checking

Formal model checking [17] consists of a systematic exhaustive exploration of all states and transitions in a model.

Model checking tools face an exponential blow up of the number of state elements (e.g. registers, latches), commonly known as the state explosion problem, which must be addressed to solve most real-world problems.

Model checking is largely automatic, being the most successful approach to formal verification in use today. Model checking tools are typically classified as equivalence checking or property checking.

2.2.1 Formal equivalence checking

Formal equivalence checking is a method of proving the equivalence of two different views of the same logic design. It uses mathematical techniques to verify equivalence of a reference design and a modified design. It is critical that the reference design is functionally correct.

2.2.2 Formal property checking

In formal property checking, properties describing the desirable/undesirable features of the design are specified using some formal logic (e.g. temporal logic) and verification is performed by proving or disproving that the property is satisfied by the model.

This work concentrates on efficient property checking techniques that make formal verification practical and realizable.

Model checking approaches are broadly classified into two types, based on state enumerations techniques employed: explicit and implicit (or symbolic).

- Explicit

  Explicit model checking techniques [49] store the explored states in a large hash table, where each entry corresponds to a single system state. A system with as few as hundreds state elements amounts to a state space with 1011 states, arriving easily to the state explosion problem.

- Implicit (symbolic)

  Symbolic model checking techniques [57] stores sets of explored states symbolically by using characteristics functions represented by canonical/semi-canonical structures, and traverse the state space symbolically by exploring a set of states in a single operation.

Symbolic model checking will be largely explored in the next section.

2.3 Symbolic Model Checking

Symbolic model checking is a method to prove properties about finite transition systems, using symbolic state enumeration. The properties are usually in one of the different flavors of temporal logic formulas such as Linear Time Logic (LTL) [66] or Computational Tree Logic (CTL) [12; 23].

According to Figure 2.2, this section will cover model checking techniques based on three main tasks [34]: finding counter-examples or bugs (Falsification layer), proving the correctness of the specification (Proof Methods layer) and obtaining smaller models that make verification possible (Abstraction layer). Prior to that, basic infrastructure required to build scalable verification algorithms are presented (Infrastructure layer).

As it will be seen, infrastructure layer also implements different heuristics and enhancements that can change general performance of formal verification.

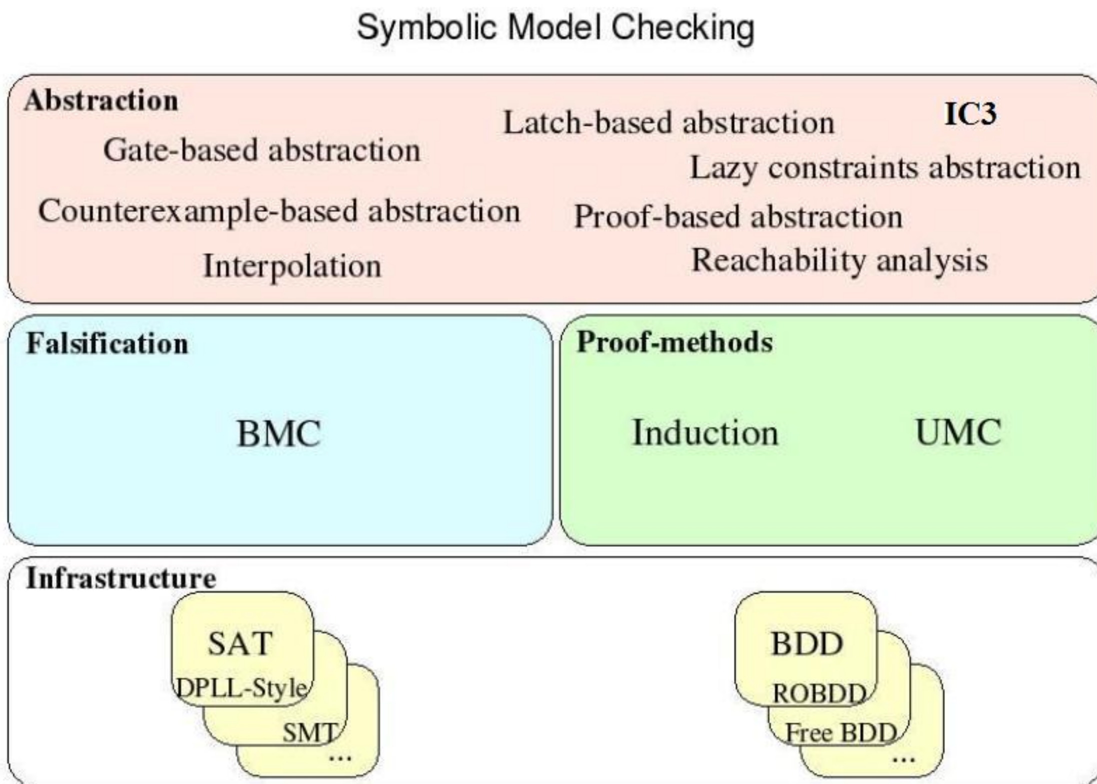

Figure 2.2. Symbolic model checking techniques overview.

2.3.1 Infrastructure Layer

Canonical structures as Binary Decision Diagrams (BDDs) [7; 19] allow constant time satisfiability checks of Boolean expressions and are used to perform symbolic Boolean function manipulations [7; 77]. Although BDD-based methods have greatly improved scalability in comparison to explicit state enumeration techniques, they are still limited to designs with a few hundred state holding elements.

Research has been heavily aimed at separating Boolean reasoning and representation. Boolean Satisfiability (SAT), which has been studied over several decades, has emerged [82] as a strong potential for Boolean reasoning, primarily due to recent advances in Davis-Putnam-Longemann-Loveland-style (DPLL-style) [26] SAT-solvers [56; 62; 41; 38]. Efficient Boolean representations such as semi-canonical representations, that are simple and reduced, are also emerging as a "de facto" structure due to their smaller sensitivity to variable ordering and compact representations compared to BDDs.

2.3.1.1 Binary Decision Diagrams (BDDs)

A BDD is just a data structure for representing a Boolean function. Bryant [19] introduced the BDD in its current form, although the general ideas have been around for quite some time.

A BDD is a directed acyclic graph in which each vertex is labeled by a Boolean variable and has outgoing arcs labeled as "then" and "else" branches. The value of the function for a given assignment to the inputs is determined by traversing from the root down to a terminal label, each time following the then (else) branch corresponding to the value 1 (0) assigned to the variable specified by the vertex label. The value of the function then equals the terminal value.

In an ordered BDD, all vertex labels must occur according to a total ordering of the variables. In a reduced ordered BDD (ROBDD), besides the ordering criteria, two more conditions are imposed:

1. two nodes with isomorphic BDDs are merged, and

2. any node with identical children is removed.

These conditions make a reduced ordered BDD a canonical representation for Boolean functions. In the sequel, BDD will refer to ROBDD.

Conceptually, we can construct the BDD for a Boolean function as follows. First build a decision tree for the desired function, obeying the restrictions that along any path from root to leaf, no variable appears more than once, and that along every path from root to leaf, the variables always appear in the same order, as it can be seen in Figure 2.3.

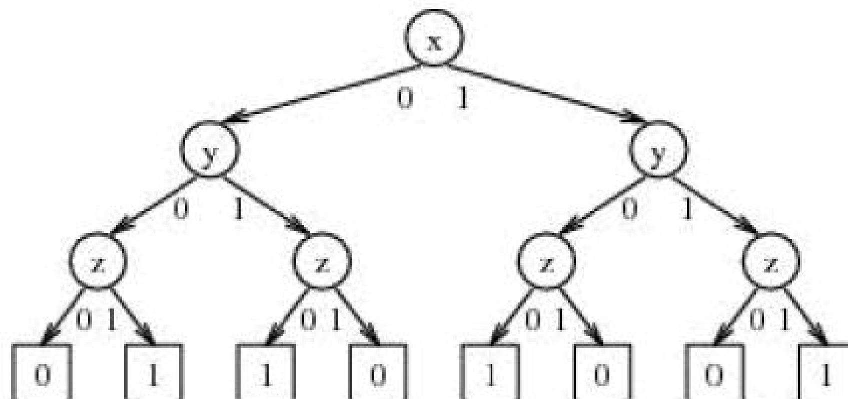

Figure 2.3. Decision tree diagram.

Next, apply the following two reduction rules as much as possible: (1) merge any duplicate (same value and same children) nodes, and (2) if both child pointers of a node point to the same child, delete the node because it is redundant, as it can be seen in Figure 2.4.

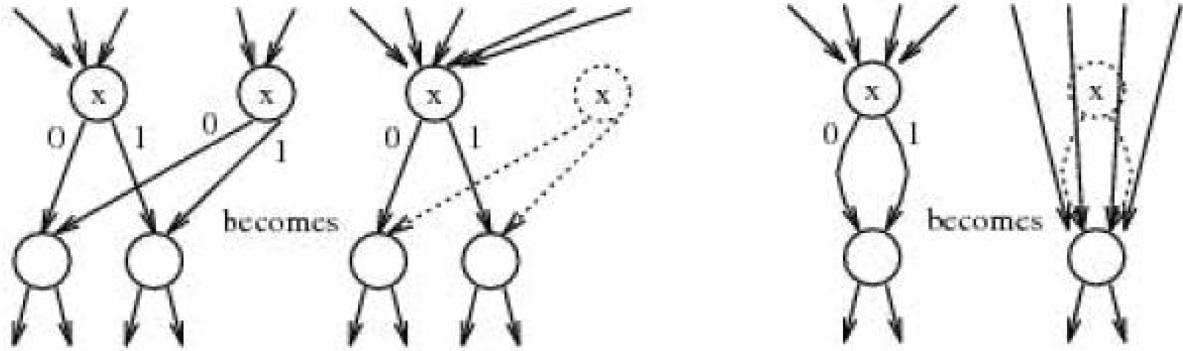

Figure 2.4. Reductions: merge duplicates and eliminate redundancy.

The resulting directed, acyclic graph, presented in Figure 2.5, is the BDD for the function.

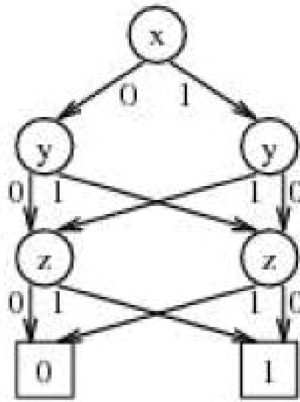

Figure 2.5. Final BDD.

In practice, BDDs are generated and manipulated in the fully reduced form, without ever building the decision tree. In a typical implementation, all BDDs in use by an application are merged as much as possible to maximize node sharing, so a function is represented by a pointer to its root node.

Once we fix the order in which the variables appear, a BDD is a canonical representation for a Boolean function. Thus, comparing Boolean functions becomes just a pointer comparison.

Choosing a good variable order is important. In general, the choice of variable order can make the difference between a linear size BDD and an exponential one.

Researchers have developed several heuristics to obtain a good variable ordering that produce compact BDD representation [10; 31; 71]. Even though finding a good BDD variable ordering is not easy [16], for some functions such as integer multiplication, there does not exist an ordering that gives a sub-exponential size representation [20].

The primary limitation of the BDD-based approaches is the scalability, i.e., BDDs constructed in the course of verification often grow extremely large, resulting in space-outs or severe performance degradation due to the paging [68]. Moreover, BDDs are not very good representations of state sets, especially when the sharing of the nodes is limited. Choosing a right variable ordering for obtaining compact BDDs is very important. Finding a good ordering is often time consuming and/or requires good design insight, which is not always feasible. Several variations of BDDs such as Free BDDs [48], zBDDs [61], partitioned-BDDs [64] and subset-BDDs [70] have also been proposed to target domain specific application; however, in practice, they have not scaled adequately for industry applications. Therefore, BDD-based approaches are limited to designs of the order of a few hundred state holding elements; this is not even at the level of an individual designer subsystem.

2.3.1.2 Boolean Satisfiability (SAT)

The Boolean Satisfiability (SAT) problem is a well-known constraint satisfaction problem, with many applications in the fields of VLSI Computer-Aided Design (CAD) and Artificial Intelligence. Given a propositional formula, the Boolean Satisfiability problem is to determine, whether there exists a variable assignment under which the formula evaluates to true, or to prove that no such assignment exists. The SAT problem is known to be NP-Complete [40]. In practice, there has been tremendous progress in SAT solver technology over the years, summarized in a survey [82].

Most SAT solvers use a Conjunctive Normal Form (CNF) representation of the Boolean formula. In CNF, the formula is represented as a conjunction of clauses, each clause is a disjunction of literals, and a literal is a variable or its negation. Note that in order for the CNF formula to be satisfied, each clause must also be satisfied, i.e., at least one literal in the clause has to be true. Converting a (gate-level) netlist to a CNF formula is straightforward: each gate translates to a set of at most three clauses. For example, an AND gate $a = b\&c$ defines the relation $a => b\&c$ and $b\&c => a$, which gives the clauses $(!a + b)$, $(!a + c)$ and $(a+!b+!c)$.

The earliest SAT algorithm was developed by Davis and Putnam [27]. Their

algorithm iteratively selects variables to resolve until no more variables are left (the problem is satisfiable) or a conflict is encountered (the problem is unsatisfiable). This is equivalent to the existential quantification of variables, and is exponential in memory usage.

Later, Davis, Longemann and Loveland [26] proposed a Depth-First Search (DFS)-based algorithm with backtracks. For historical reasons it is also known as the Davis-Putnam-Longemann-Loveland (DPLL) algorithm. Most modern high-performance SAT solvers are based on a DPLL-style procedure.

A typical SAT solver relies on a few key procedures:

- Decides: makes a decision by picking an unassigned variable and assigning it a value (0 or 1). It returns false if there are no more unassigned variables.
- Deduces: propagates the decision assignment and assignments implied by propagation in the clauses. It returns false if these assignments conflict.
- Analyses: determines, when a conflict occurs, which decision to backtrack. Optionally it learns new information from the conflict.
- Backtrack: undoes assignments up to a specified decision level. A new assignment is made to branch the search.

Algorithm below shows how these procedures work together.

```
SAT() {
    Dlevel = 0;
    While (true) {
        If (deduce() == false) {
            Dlevel = analyse();
            If (dlevel == 0) return UNSAT;
            Else backtrack(dlevel);
        }
        Else if (decide() == false) return SAT;
        Else dlevel = dlevel + 1;
    }
}
```

As simple as it looks, this algorithm has been the subject of continuing research for the past forty years. In fact, each of the main procedures offers unique opportunity

for improvement. The order in which a variable is picked, and the value assigned in the decision, can affect the actual space. The same is true with conflict analysis and backtracking. Backtracks can be made non-chronologically (that is, not just backtrack to the last decision) with an intelligent analysis of the conflict. Also, the efficient implementation of the heavily used deduction/propagation procedure can greatly improve the overall performance of the algorithm.

Examples of SAT solvers are CHAFF [62], GRASP [74] and SATO [80].

Algorithm advances are the key to the success of SAT-based formal methods. Some of the most promising results on certain problem instances that involve low overhead will be now mentioned.

Frequent restarts The state-of-the-art SAT solvers also employ a technique called random restart [62] for greater robustness. The first few decisions are very important in the SAT solver. A bad choice could make it very hard for the solver to exit a local non-useful search space. Since it is very hard to decide a priori what a good choice might be for decisions, the restart mechanism periodically undoes all decisions and starts afresh. The learned clauses are preserved between restarts; therefore, the search conducted in previous rounds is not lost. By utilizing such randomizations, a SAT solver can minimize local fruitless search.

Non-conflict-driven Back-jumping This refers to a back-jump to an earlier decision level (not necessarily to level 0), without detecting a conflict [65]. It is a variation of frequent restarts strategy, but guided by the number of conflict-driven backtracks seen so far. The goal is to quickly get out of a "local conflict zone" when the number of backtracks occurring between two decision levels exceeds a certain threshold. For hard problem instances, such strategy has shown promising results.

Frequent Clause Deletion Conflict-driven learned clauses are redundant, and therefore, deleting them does not affect satisfiability of the problem. Conflict clauses, though useful can become an overhead especially due to increased Boolean Constraint Propagation (BCP) time and due to large memory usage. Such clauses can be deleted based on their relevance metric [62] which is computed based on number of unassigned literals. One can also compute relevance of a clause based on its frequent involvement in conflict [41].

Clause Shrinking Effectiveness of conflict-driven learning scheme is very hard to determine a priori. In general, a shorter conflict-clause is useful, as it prunes a larger search space. One scheme is to shrink the conflict clause by identifying

a sufficient subset of the literals required to generate the conflict [63]. Using the conflict literals as decision variables, one applies BCP and stops as soon as a conflict is detected. In many cases, fewer literals in the conflict clause are involved.

Early Conflict Detection in Implication Queue The implication queue (in lazy assignment) stores the newly implied variables during BCP. If a newly implied variable is already in the queue with an opposite implied value, conflict can be detected early without doing BCP further [14].

Shorter Reasons First Several unit clauses can imply the same value on a variable at a given decision level. With an intuition that shorter unit clauses decrease the size of implication graph, and hence, the size of conflict clauses, the implications due to shorter clauses are given preference [14].

Satisfiability Modulo Theory (SMT) An SMT instance is a generalization of a Boolean SAT instance in which various sets of variables are replaced by predicates from a variety of underlying theories. Obviously, SMT formulas provide much richer modeling language than is possible with Boolean SAT formulas. For example, an SMT formula allow us to model the data path operations of a microprocessor at the word rather than the bit level.

Formally speaking, an SMT instance is a formula in quantifier-free first-order logic, and SMT is the problem of determining whether such formula is satisfiable. In other words, imagine an instance of the Boolean satisfiability problem (SAT) in which some of the binary variables are replaced by predicates over a suitable set of non-binary variables. A predicate is basically a binary-valued function of non-binary variables. Example of predicates include linear inequalities (e.g., $3x+2y-z > 4$) or equalities involving so-called uninterpreted terms and function symbols (e.g., $f(f(u,v),v) = f(u,v)$ where f is some unspecified function of two unspecified arguments). These predicates are classified according to the theory they belong to. For instance, linear inequalities over real variables are evaluated using rules of the theory of linear real arithmetic, whereas predicates involving uninterpreted terms and function symbols are evaluated using rules of the theory of uninterpreted functions with equality (sometimes referred to as the empty theory).

Early attempts for solving SMT instances involved translating them to Boolean SAT instances (e.g., a 32-bit integer variable would be encoded by 32 bit variables with appropriate weights and word-level operations such as plus would be

replaced by lower-level logic operations on the bits) and passing this formula to a Boolean SAT solver. This approach has its merits: by pre-processing the SMT formula into an equivalent Boolean SAT formula we can use existing Boolean SAT solvers "as-is" and leverage their performance and capacity improvements over time. On the other hand, the loss of the high-level semantics of the underlying theories means that SAT solver has to work a lot harder than necessary to discover obvious facts (such as $x + y = y + x$ for integer addition). This observation led to the development of a number of SMT solvers that tightly integrate the Boolean reasoning of a DPLL-style search with theory-specific solvers that handle conjunctions (ANDs) of predicates from a given theory.

Dubbed DPLL(T) [39], this architecture gives the responsibility of Boolean reasoning to the DPLL-based SAT solver which, in turn, interacts with a solver for theory T through a well-defined interface. The theory solver needs only to worry about checking the feasibility of conjunctions of theory predicates passed on to it from SAT solver, as it explores the Boolean search space of the formula. For this integration to work well, however, the theory solver must be able to participate in propagation and conflict analysis.

2.3.2 Falsification and Proof Methods Layer

Falsification and proof methods, as presented in Figure 2.2, are two opposite approaches, to prove that a property holds or does not hold for a model. Falsification methods cannot verify properties, just falsify them.

2.3.2.1 Bounded Model Checking (BMC)

BMC has been gaining ground as a falsification engine, mainly due to its improved scalability compared to other formal techniques.

In BMC, the focus is on finding a counterexample (CEX) - bug, of a bounded length k. For a given design and correctness property, the problem is translated effectively to a propositional formula such that the formula is true if and only if a counterexample of length k exists [15]. If no such counterexample is found, k is increased. This process terminates when k exceeds the completeness threshold - CT (i.e., k is sufficiently large to ensure that no counterexample exists) - or when SAT procedure exceeds its time or memory bounds.

Such a translation basically involves unrolling the circuit of the transition relation for the required number of time frames. Essentially, k copies of the circuit are made

and then clauses are build at each time frame for the unrolled circuit and the property to be checked, which is then fed to a SAT-solver.

Many enhancements have been proposed in the last few years to make the standard BMC procedure [15] scale with large industry designs. One key improvement is dynamic circuit simplification [32], performed on the iterative array model of the unrolled transition relation, where an on-the-fly circuit reduction algorithm is applied not only within a single time frame but also across time frames to reduce the associated Boolean formula. Another enhancement is SAT-based incremental learning, used to improve the overall verification time by re-using the SAT results from the previous runs [32]. Learning can also be accomplished by a lightweight and goal-directed effective BDD-based scheme, where learned clauses generated by BDD-based analysis are added to the SAT solver on-the-fly, to supplement its other learning mechanisms [46]. There are also many heuristics for guiding the SAT search process to improve the performance of the BMC engine.

Even with the many enhancements just mentioned, sometimes the memory limitation of a single server, rather than time, can become a bottleneck for doing deeper BMC search on large designs. The main limitation of a standard BMC application is that it can perform search up to a maximum depth allowed by the physical memory on a single server. This limitation stems from the fact that as the search bound k becomes large, the memory requirement due to unrolling of the design also increases. Especially for memory-bound designs, a single server can quickly become a bottleneck in doing deeper search for bugs. Distributing the computing requirements of BMC (memory and time) over a network of workstations can help overcome the memory limitation of a single server, albeit at an increased communication cost [37].

2.3.2.2 Induction

Although BMC can find bugs in larger designs than BDD-based methods, the correctness of a property is guaranteed only for the analysis bound. However, one can augment BMC for performing proofs by induction [73]. A completeness bound has been proposed [15] to provide an inductive proof of correctness for safety properties based on the longest loop-free path between states. Induction with increasing depth k, and restriction to loop-free paths, consists of the following two steps:

- Base: to prove that the property holds on every k-length path starting from the initial state.

- k-step Induction: to prove that if the property holds on a k-length path starting from an arbitrary state, then it also holds on all its extensions to a $(k+1)$-length path.

Algorithm below and Figure 2.6 illustrate k-induction.

```
Procedure k-induction(M, p)
1. initialize k = 0
2. while true do
3.    if Base(M, p, k) is SAT
4.    then return counterexample
5.    else
6.         if Step(M, p, k) is UNSAT
7.         then return verified
8.    k = k + 1
9. end while
end
```

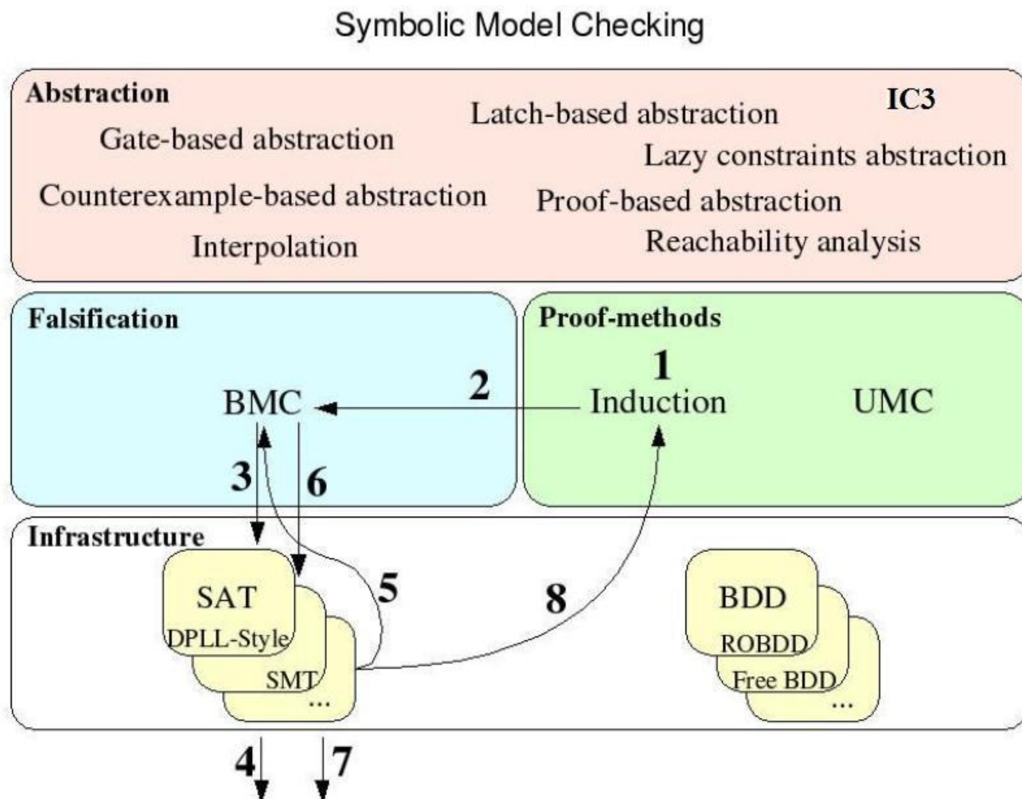

Figure 2.6. Graphical visualization of k-induction algorithm.

The restriction to loop-free paths imposes the additional constraints that no two states in the paths are identical. Note that the base case includes use of the initial state constraint, but the inductive step does not. Therefore, the inductive step may include unreachable states also. In practice, this may not allow the induction proof to go through without the use of additional constraints, i.e., stronger induction invariants than the property itself. To shorten the proof length, one can use any circuit constraints known by the designers as inductive invariants.

A set of over-approximate reachable states of the designs can be regarded as providing reachability constraints. These can be used as inductive invariants to strengthen a proof by induction [45]. In principle, any technique can be used to obtain such over-approximate reachability constraints, including information known by the designer.

2.3.2.3 Unbounded Model Checking (UMC)

UMC comprises methods that can prove the correctness of a property on a design as well as find counter-examples for failing properties.

BDD-based model checking tools do not scale well with design complexity and size. SAT-based BMC tools provide faster counter-example checking, but a proof may require unrolling of the transition relation up to the longest loop-free path. Previous approaches to SAT-based unbounded model checking suffer from large time and space requirements for solution enumeration, and hence are not viable.

New efficient and scalable approaches have been created, like the one based on circuit co-factoring for SAT-based quantifier elimination [35], that dramatically reduces the number of required enumeration steps, thereby, significantly improving the performance of pre-image and fixed-point computation in SAT-based UMC. The circuit cofactoring method uses Reduced AIG (AND/INVERTER Graph) representation for the state sets as compared to BDDs and CNF representations. The novelty of the method is in the use of circuit co-factoring to capture a large set of states, i.e., several state cubes in each SAT enumeration step, and in the use of circuit graph simplification based on functional hashing to represent the captured states in a compact manner.

SAT-based BMC can be augmented with unbounded SAT-based analysis to obtain the longest shortest diameter as the completeness bounds for these properties, instead of using loop-free path analysis. Inductive invariants such as reachability constraints can be used to shorten completeness bound further.

2.3.3 Abstraction Layer

Obtaining appropriate abstract models that are small and suitable for applying falsification or proof methods have been the subject of research for quite some time [24; 54].

Abstraction means, in effect, removing information about a system which is not relevant to a property to be verified. In the simplest case, a system can be viewed as a large collection of constraints, and abstraction as removing constraints that are irrelevant. The goal in this case is not so much to eliminate constraints per se, as to eliminate state variables that occur only in irrelevant constraints, and thereby to reduce the size of the state space. A reduction of the state space in turn increases the efficiency of model checking, which is based on exhaustive state space exploration.

Figure 2.2 lists some well known abstractions, which are described below.

2.3.3.1 Gate-based abstraction

Gate-based abstraction is a basic approach to reduce the number of gates to consider during verification. Basically, after a property has been selected to be proved, just the gates that are important to the proof will be considered.

For a signal s, representing the property to be proved, just the transitive fanin of signal s should be considered. The transitive fanin of a signal s is the set of gates that transitively drives the signal s through some other gates (not registers). The transitive fanin arrives to the primary inputs of the circuit that may influence the value of signal s. The gates in the path from these primary inputs up to signal s are the gates selected by gate-based abstraction approach.

2.3.3.2 Latch-based abstraction

Consider a SAT problem is unsatisfiable at a given depth k, i.e., there is no counterexample for the safety property at a given depth k. Latch-based abstraction can be used to obtain a smaller size abstract model. Rather than optimize at the level of each gate in the original design, a latch-based abstraction technique is targeted to minimize the set of latch reasons at depth k, while still retaining the useful property that there is no counterexample of depth k. An abstract model is then generated for depth k by converting those latches in the given design that are not in the set of latch reasons to pseudo-primary inputs, PPI (pseudo-primary inputs refer to output signals of those latches whose next state logic is removed and the signals behaves as primary inputs). Depending on the locality of the property, the set of latch reasons can be significantly smaller than the total latches in the given design.

2.3.3.3 Lazy constraints

Consider a SAT problem is unsatisfiable at a given depth k, i.e., there is no counterexample for the safety property at a given depth k. The following partition/classification of the set of latches is valid:

- Propagation latches: for which at least one interface propagation constraint belongs to set of latch reasons.
- Initial value latches: for which only an initial state constraint belongs to set of latch reasons.
- PPI latches: for which neither the initial constraint, nor any of the interface propagation constraints belongs to set of latch reasons.

Clearly, the set of PPI latches can be abstracted away, since they are not used at all in the proof of unsatisfiability (this has been done in the latch-based abstraction). On the other hand, a propagation latch needs to be retained in the abstract model, since it was used to propagate a latch constraint across time frames for the derived proof. The more interesting case is presented by an initial value latch. It is quite possible that an initial value latch is not really needed to derive unsatisfiability - its initial state constraint may just happen to be used by the SAT solver. It has been observed empirically [34] that on large designs, a significant fraction (as high as 20% in some examples) of the marked latches are initial value latches. Rather than add these latches to an abstract model, the strategy is to guide the SAT solver to find a proof that would not use their initial state values unless needed. This is done by the use of lazy constraints.

A naive way of delaying implications due to initial state constraints is to mark the associated variables, and delay BCP (Boolean Constraint Propagation) on these marked variables during pre-processing. However, this would involve an overhead of checking for such variables during BCP. Rather than change standard BCP procedure, the desired effect can be achieved by changing the CNF representation of these constraints. This allows the exploration of the latest improvements in SAT solver technology without modifying the SAT solvers.

2.3.3.4 Counterexample-based abstraction

Counterexample-based abstraction-refinement [54] is an iterative technique that starts with BDD-based UMC on an initial conservative abstraction of the model. If UMC proves the property on the abstraction then the property is true on the full model.

However, if a counterexample A is found, it could either be an actual error or it may be spurious, in which case one needs to refine the abstraction to rule out this counterexample. The process is then repeated until the property is found to be true, or until a real counterexample is produced.

Algorithm below and Figure 2.7 illustrate CEX-based abstraction.

```
Procedure cex-based (M, p)
1. generate initial abstraction M'
2. while true do
3.    if UMC(M', p) holds
4.    then return verified
5.    let k = length of abstract counterexample A
6.    if BMC(M, p, k, A) is SAT
7.    then return counterexample
8.    else use proof of UNSAT P to refine M'
9. end while
end
```

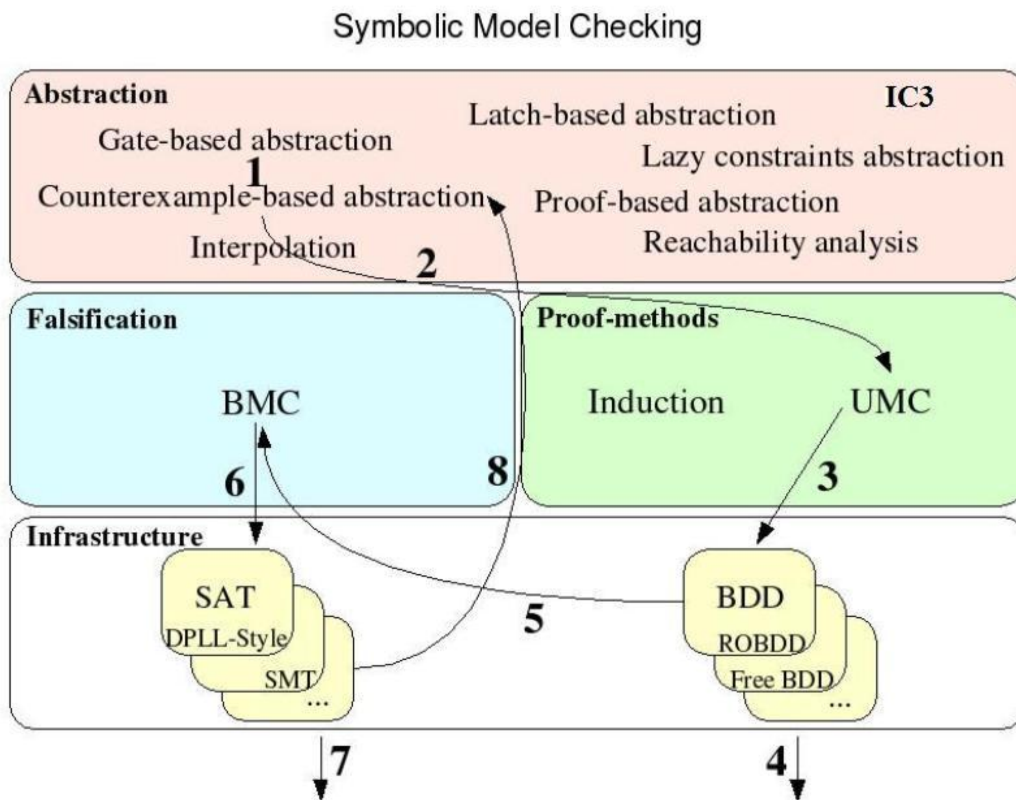

Figure 2.7. Graphical visualization of CEX-based abstraction algorithm.

2.3.3.5 Proof-based abstraction

The proof-based algorithm in [60] iterates through SAT-based BMC and BDD-based UMC. It starts with a short BMC run, and if the problem is satisfiable, an error has been found. If the problem is unsatisfiable, the proof of unsatisfiability is used to guide the formation of a new conservative abstraction on which BDD-based UMC is run. In case that the BDD-based model checker proves the property then the algorithm terminates; otherwise the length k' of the counterexample generated by the model checker is used as the next BMC length.

Notice that only the length of the counterexample generated by BDD-based UMC is used. This method creates a new abstraction each iteration, in contrast to the counterexample abstraction method which refines the existing abstraction.

Since this abstraction includes all variables in the proof of unsatisfiability for a BMC run up to depth k, it is known that any counterexample obtained from model checking this abstract model will be of length greater than k. Therefore, unlike the counterexample method, this algorithm eliminates all counterexamples of length k in a single unsatisfiable BMC run.

This procedure, shown below, continues until either a failure is found in the BMC phase or the property is proved in the BDD-based UMC. The termination of the algorithm hinges in the fact that the value k' increases in every iteration. Figure 2.8 illustrates proof-based abstraction.

```
Procedure proof-based (M, p)
1. initialize k
2. while true do
3.    if BMC(M, p, k) is SAT
4.    then return counterexample
5.    else
6.        drive new abstraction M’ from proof P
7.        if UMC (M’, p) holds
8.        then return verified
9.        else set k to length of counterexample k’
10. end while
end
```

Note that the abstract model obtained using proof-based abstraction technique is property-specific, i.e., each property may lead to a different abstraction.

Proof-based abstraction technique has many other implementations ([44; 36; 21]).

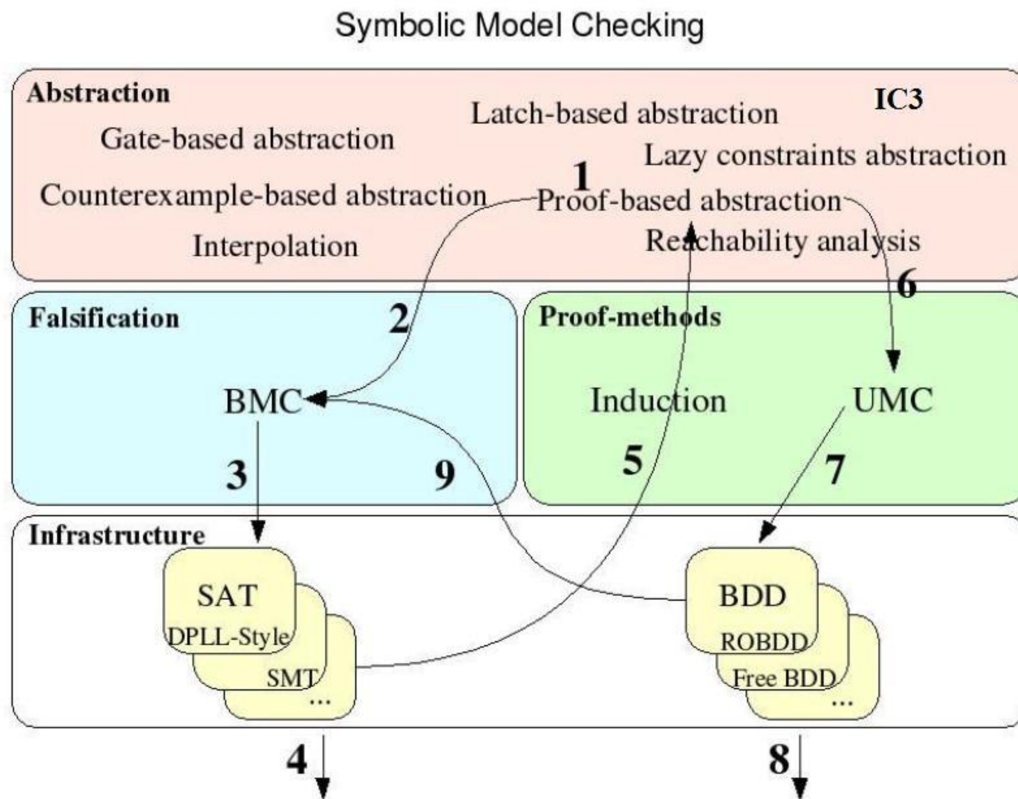

Figure 2.8. Graphical visualization of Proof-based abstraction algorithm.

2.3.3.6 Reachability constraints

Given a finite state machine (FSM) of a hardware design, a state t is said to be reachable from s if there is a sequence of transitions that start from s and ends at t.

BDD-based symbolic traversal techniques to perform a reachability analysis on the abstract model can be used to compute an over-approximate reachable set for the concrete design. These are used as additional reachability constraints during proofs by induction or SAT-based Unbounded Model Checking.

2.3.3.7 Interpolation

An interpolant I for an unsatisfiable formula $A \wedge B$ is a formula such that:

1. $A \rightarrow I$
2. $I \wedge B$ is unsatisfiable and
3. I refers only to the common variables of A and B.

Intuitively, I is the set of facts that the SAT solver considers relevant in proving the unsatisfiability of $A \wedge B$.

The interpolation-based algorithm [59] uses interpolants to derive an over-approximation of the reachable states with respect to the property. The BMC problem $BMC(M, p, k)$ is solved for an initial depth k. If the problem is satisfiable, a counterexample is returned, and the algorithm terminates. If $BMC(M, p, k)$ is unsatisfiable, the formula representing the problem is partitioned into $Pref(M, p, k) \wedge Suff(M, p, k)$, where $Pref(M, p, k)$ is the conjunction of the initial condition and the first transition, and $Suff(M, p, k)$ is the conjunction of the rest of the transitions and the final condition. The interpolant I of $Pref(M, p, k)$ and $Suff(M, p, k)$ is computed. Since $Pref(M, p, k) \rightarrow I$, it follows that I is true in all states reachable from $I(s0)$ in one step. This means that I is an over-approximation of the set of states reachable from $I(s0)$ in one step. Also, since $I \wedge Suff(M, p, k)$ is unsatisfiable, it also follows that no state satisfying I can reach an error in $k - 1$ steps. If I contains no new states, that is, $I \rightarrow I(s0)$, then a fixed point of the reachable set of states has been reached, thus the property holds. I I has new states then R' represents an over-approximation of the states reached so far. The algorithm then uses R' to replace the initial set I, and iterates the process of solving the BMC problem at depth k and generating the interpolant as the over-approximation of the set of states reachable in the next step. The property is determined to be true when the BMC problem with R' as the initial condition is unsatisfiable, and its interpolant leads to a fixed point of reachable states. However, if the BMC problem is satisfiable, the counterexample may be spurious since R' is an over-approximation of the reachable set of states. In this case, the value of k is increased, and the procedure is continued. The algorithm will eventually terminate when k becomes larger than the diameter of the model. Figure 2.9 illustrates interpolation.

```
Procedure interpolation (M, p)
1. initialize k
2. while true do
3.    if BMC(M, p, k) is SAT
4.    then return counterexample
5.    R = I
6.    while true do
7.        M' = (S,R,T,L); let C = Pref(M',p,k) and Suff(M',p,k)
8.        if C is SAT
9.        then break (go to line 13)
```

```
10.         compute interpolant I of C;
            R’ = I is an over-approximation of states reachable
                 from R in one step
11.         if R => R’ then return verified
            R = R or R’
12.     end while
13.     increase k
14. end while
end
```

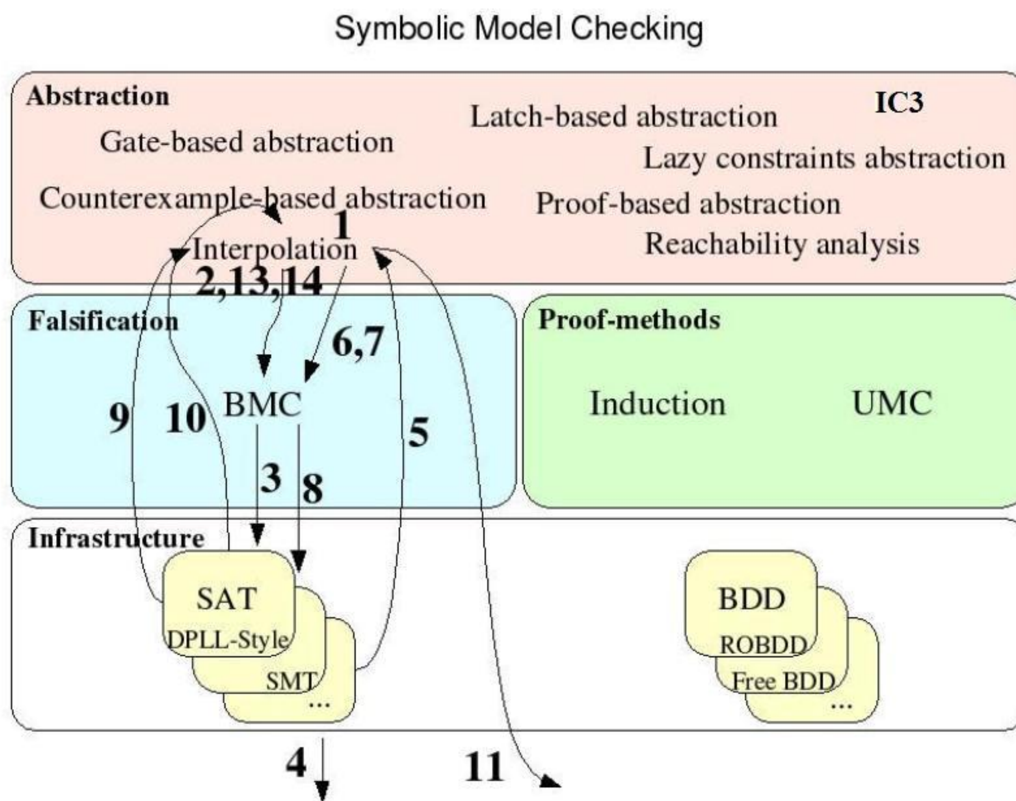

Figure 2.9. Graphical visualization of interpolation algorithm.

2.3.4 Proof engines

Many heuristics to accomplish formal verification have been presented so far. Each group of algorithms and configuration parameters generates a different solver or engine. The reason for having a large number of engines relies on the fact that verification performance varies from design to design and from property to property. Mixing heuristics increases engine’s scalability and efficiency.

Since users have limited resources for the verification of systems, it is important to know which, of the huge number of available engines, are most effective. Imagine we take 5 different abstraction algorithms, together with 26 SAT solvers (26 is the number of SAT solvers competitors registered in the SAT Competition 2011 - $http://www.cril.univ-artois.fr/SAT11/$) and 1 BDD solver. The number of engines we get is 135 $(5 * (26 + 1))$. This shows that human intuition of the best engines to run becomes not so straightforward and selecting the best engines in an intelligent manner becomes a differential.

But, as we know, the performance of an engine is strongly dependent on circuit design and property to be proved. So a basic problem arises: how to decide which engines to use for each specific circuit design and property, in order to maximize the chance of reaching a proof conclusion?

Chapter 3

Statistical Learning

Statistical learning plays a key role in many areas of science, finance and industry. Here are some examples of learning problems [47]:

- Predict whether a patient, hospitalized due to a heart attack, will have a second heart attack. The prediction is to be based on demographic, diet and clinical measurements for that patient.

- Predict the price of a stock in 6 months from now, on the basis of company performance measures and economic data.

- Identify the numbers in a handwritten ZIP code, from a digitized image.

- Estimate the amount of glucose in the blood of a diabetic person, from the infrared absorption spectrum of that person's blood.

- Identify the risk factors for prostrate cancer, based on clinical and demographic variables.

The science of learning plays a key role in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines.

In a typical scenario, we have an outcome measurement, usually quantitative (like a stock price) or categorical (like heart attack/no heart attack), that we wish to predict based on a set of features (like diet and clinical measurements). We have a training set of data, in which we observe the outcome and feature measurements for a set of objects (such as people). Using this data we build a prediction model, or learner, which will enable us to predict the outcome for new unseen objects. A good learner is one that accurately predicts such an outcome.

The objective of supervised learning is to use the inputs to predict the values of the outputs. The inputs, that are measured or preset, constitute the set of variables. They have some influence on one or more outputs. In the statistical literature the inputs are often called the predictors, a term that will be used interchangeably with inputs, and more classically the independent variables. The outputs are called the responses, or classically the dependent variables.

The outputs vary in nature. They may be quantitative measurements or qualitative values from a finite set. For both types of outputs it makes sense to think of using the inputs to predict the output. This distinction in output type has led to a naming convention for the prediction tasks: regression, when we predict quantitative outputs, and classification, when we predict qualitative outputs. These two tasks have a lot in common, and in particular both can be viewed as a task in function approximation.

3.1 Multivariate regression

When fitting function for experimental data modeling have more then one independent argument we can talk about multivariate regression.

Despite the recent progress in statistical learning, nonlinear function approximation with high-dimensional input data remains a nontrivial problem.

An ideal algorithm for such tasks needs to:

- avoid potential numerical problems from redundancy in the input data,
- eliminate irrelevant input dimensions,
- keep the computational complexity of learning updates low while remaining data efficient,
- allow for on-line incremental learning, and
- achieve accurate function approximation and adequate generalization.

Linear models were largely developed in the pre-computer age of statistics, but even in today's computer era there are still good reasons to study and use them. They are simple and often provide an adequate and interpretable description of how the inputs affect the output. For prediction purposes they can sometimes outperform fancier nonlinear models, especially in situations with small numbers of training cases, low signal-to-noise ratio or sparse data. Finally, linear methods can be applied to transformations of the inputs and this considerably expands their scope.

Principal component analysis (PCA) is an important methodology that should be used when multivariate regression is in use. Its main objective is to find patterns in the input data, allowing the reduction of the number of dimensions without much information loss [75].

More recently k-nearest neighbors method has also been applied in conjunction with linear regression in different fields like semiconductor manufacturing.

3.2 Linear regression

The linear model has been a mainstay of statistics for the past 30 years and remains one of our most important tools.

There are many different methods to fit the linear model to a set of training data, but by far the most popular is the method of linear least squares. Used directly, with an appropriate data set, linear least squares regression can be used to fit the data with any function of the form

$$f(\vec{x};\vec{b}) = b_0 + b_1x_1 + b_2x_2 + ... \tag{3.1}$$

in which

1. each explanatory (independent) variable x_i in the function is multiplied by an unknown parameter,

2. there is at most one unknown parameter with no corresponding explanatory variable, and

3. all of the individual terms are summed to produce the final function value.

In statistical terms, any function that meets these criteria would be called a "linear function". The term "linear" is used, even though the function may not be a straight line, because if the unknown parameters are considered to be variables and the explanatory variables are considered to be known coefficients corresponding to those "variables", then the problem becomes a system (usually overdetermined) of linear equations that can be solved for the values of the unknown parameters. To differentiate the various meanings of the word "linear", the linear models being discussed here are often said to be "linear in the parameters" or "statistically linear".

For example, linear models are not limited to being straight lines or planes, but include a fairly wide range of shapes. A simple quadratic curve

$$f(x;\vec{b}) = b_0 + b_1 x + b_{11} x^2 \tag{3.2}$$

is linear in the statistical sense. A straight-line model in $\log(x)$

$$f(x;\vec{b}) = b_0 + b_1 \ln(x) \tag{3.3}$$

or a polynomial in $\sin(x)$

$$f(x;\vec{b}) = b_0 + b_1 \sin(x) + b_2 \sin(2x) + b_3 \sin(3x) \tag{3.4}$$

are also linear in the statistical sense because they are linear in the parameters, though not with respect to the observed explanatory variable.

Just as models that are linear in the statistical sense do not have to be linear with respect to the explanatory variables, nonlinear models can be linear with respect to the explanatory variables, but not with respect to the parameters. For example,

$$f(x;\vec{\beta}) = \beta_0 + \beta_0 \beta_1 x \tag{3.5}$$

is linear in x, but it cannot be written in the general form of a linear model presented above. This is because the slope of this line is expressed as the product of two parameters.

Linear least squares regression also gets its name from the way the estimates of the unknown parameters are computed. In the least squares method the unknown parameters are estimated by minimizing the sum of the squared deviations between the data and the model. The minimization process reduces the overdetermined system of equations formed by the data to a sensible system of P equations in P unknowns (where P is the number of parameters in the functional part of the model). This new system of equations is then solved to obtain the parameter estimates.

Nonlinear least squares regression could be used to fit this model, but linear least squares cannot be used.

In our work, we have:

$$f(x;\vec{b}) = \prod_i x_i^{b_i} \tag{3.6}$$

Then, it can be linearized via:

$$\log f(x;\vec{b}) = \sum_i b_i \log(x_i) \tag{3.7}$$

Or, if we have:

$$f(x;\vec{b}) = \prod_i b_i^{x_i} \tag{3.8}$$

It is linearized as:

$$\log f(x;\vec{b}) = \sum_i x_i \log(b_i) \tag{3.9}$$

Equations 3.6 and 3.8 are both instances of linear regression, as shown in Equations 3.7 and 3.9 respectively.

Linear least squares regression has earned its place as the primary tool for process modeling because of its effectiveness and completeness.

Though there are types of data that are better described by functions that are nonlinear in the parameters, many processes in science and engineering are well-described by linear models. This is because either the processes are inherently linear or because, over short ranges, any process can be well-approximated by a linear model.

The estimates of the unknown parameters obtained from linear least squares regression are the optimal estimates from a broad class of possible parameter estimates under the usual assumptions used for process modeling. Practically speaking, linear least squares regression makes very efficient use of the data. Good results can be obtained with relatively small data sets.

The theory associated with linear regression is well-understood and allows for construction of different types of easily-interpretable statistical intervals for predictions, calibrations, and optimizations. These statistical intervals can then be used to give clear answers to scientific and engineering questions.

The main disadvantages of linear least squares are limitations in the shapes that linear models can assume over long ranges, possibly poor extrapolation properties, and sensitivity to outliers.

Linear models with nonlinear terms in the predictor variables curve relatively slowly, so for inherently nonlinear processes it becomes increasingly difficult to find a linear model that fits the data well as the range of the data increases. As the explanatory variables become extreme, output of the linear model will also always more extreme. This means that linear models may not be effective for extrapolating the results of a process for which data cannot be collected in the region of interest. Of course extrapolation is potentially dangerous regardless of the model type.

Finally, while the method of least squares often gives optimal estimates of the unknown parameters, it is very sensitive to the presence of unusual data points in the data used to fit a model. One or two outliers can sometimes seriously skew the results of a least squares analysis. This makes model validation, especially with respect to out-

liers, critical to obtaining sound answers to the questions motivating the construction of the model.

Figure 3.1 shows the globally (all training set is used to build the prediction model) linear nature of linear regression in a two dimensional input data scenario.

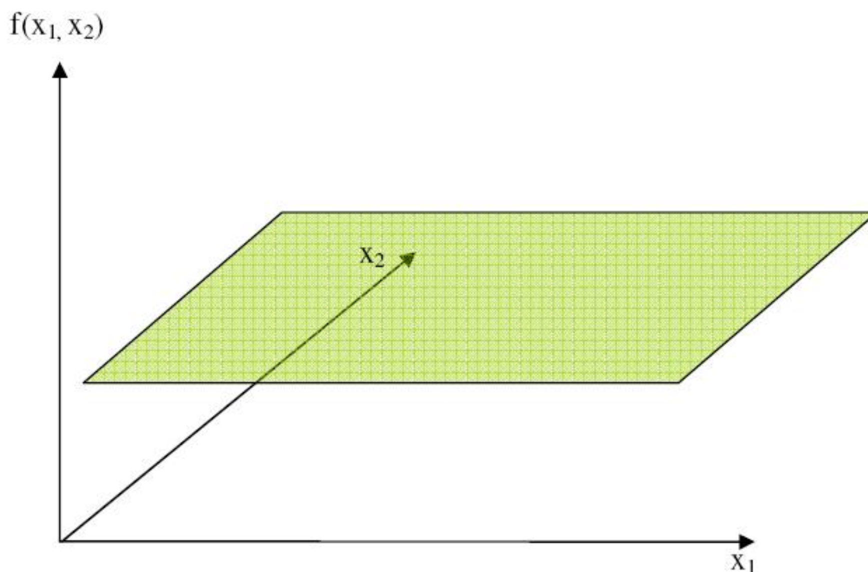

Figure 3.1. Globally linear nature of linear regression.

3.3 Nearest neighbor method

The method for data-driven modeling considered up to now was first building a model of the available (training) data (process of learning), and then was put to operation, when classification or numerical prediction was taking place. These methods are sometimes referred to as eager learning (since they are eager to build a model first). However, there is a group of learning methods in which a model is not actually constructed - such learning is called instance-based learning, or lazy learning.

Instance-based (IB) learning methods simply store training examples and postpone the generalization (building a model) until a new instance must be classified or prediction made. The model that is built by IB methods is not a global model that uses all training data, but rather a local model involving only some of the instances. The IB methods are used both for classification and for regression. Most important methods are: nearest neighbor method, locally weighted regression, and case-based reasoning.

Nearest neighbors methods use observations in the training set closest to the new instance to be classified to predict F. Specially, the k-nearest neighbors (kNN) fit for

F is defined as follows:

$$F(x_q) = \frac{\sum_{i=1}^{k} f(x_i)}{k} \tag{3.10}$$

where x_i is in the kNN neighborhood of x_q (defined by the k points closest to x_q in the training set). Closeness implies a metric which can be, for example, Euclidean distance. So, in words, we find the k observations closest to x_q in input space, and average their responses.

In order to identify neighbors in a multivariate scenario, the objects are represented by position vectors in a multidimensional feature space. It is usual to use the Euclidean distance, though other distance measures, such as the Manhattan distance could in principle be used instead. The k-nearest neighbors algorithm is sensitive to the local structure of the data.

The k-nearest neighbors algorithm is amongst the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors, with the object being assigned the class most common amongst its k nearest neighbors. k is a positive integer, typically small. If $k = 1$, then the object is simply assigned the class of its nearest neighbor. In binary (two class) classification problems, it is helpful to choose k to be an odd number as this avoids difficulties with tied votes.

Figure 3.2 shows an example for points in 2-dimensional space. The number of classes is $s = 2$, so the output is boolean (denoted as "+" or "-"). New instance x_q (called also a query point) is classified with respect to proximity of nearest training instances. If we apply 1-NN method, we will consider only 1 such training instance, and x_q will be classified to "+" (since the nearest training instance belongs to class "+"). If, however, 5-NN method is used, we consider 5 training instances, and xq will be classified to "-" (since among 5 instances there are 2 "+" and 3 "-"). This example explains the essence of the k-NN algorithm "-" to classify a new x_q it finds the most common value of the nearest training instances.

A refinement of the k-NN classification algorithm is to weigh the contribution of each of the k neighbors according to their distance to the query point x_q, giving greater weight w to closer neighbors (Weighted k-NN).

The same method can be used for regression, by simply assigning the property value for the object to be the average of the values of its k nearest neighbors.

Local weighted regression (numerical prediction) is a generalization of the nearest-neighbor approaches. It constructs an explicit approximation $F(x)$ of the target function $f(x)$ over a local region surrounding the new query point x_q. The type of this approximation can be any function: linear, quadratic, ... This method is called weighted

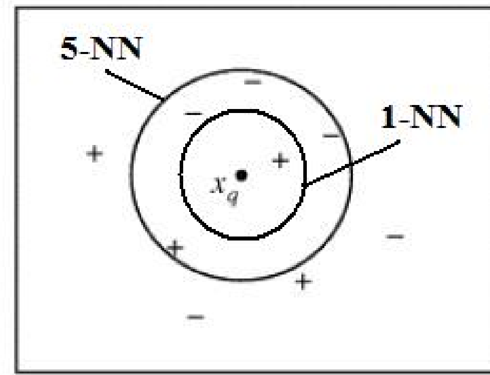

Figure 3.2. 1-NN and 5-NN neighborhood.

because the contribution of each training example is weighted by its distance to the query point x_q.

If $F(x)$ is linear then this is called locally weighted linear regression:

$$F(x) = w_0 + w_1\alpha_1(x) + ... + w_n\alpha_n(x) \tag{3.11}$$

If we construct a global model, we have to minimize the error for all training examples D:

$$E = \frac{1}{2}\sum_{x \subset D}(f(x) - F(x))^2 \tag{3.12}$$

However, for local models it is necessary to look only at the proximity of x_q. In this case there are the following possibilities:

1. Minimize the squared error over just k nearest neighbors:

$$E_1(x_q) = \frac{1}{2}\sum_{x \subset k\, nearest\, neighbors\, of\, x_q}(f(x) - F(x))^2 \tag{3.13}$$

2. Minimize the squared error over entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from x_q:

$$E_2(x_q) = \frac{1}{2}\sum_{x \subset D}(f(x) - F(x))^2 K(d(x_q, x)) \tag{3.14}$$

3. Combine E_1 and E_2 (to reduce computational costs):

$$E_3(x_q) = \frac{1}{2} \sum_{x \subset k\ nearest\ neighbors\ of\ x_q} (f(x) - F(x))^2 K(d(x_q, x)) \qquad (3.15)$$

The problem of minimization of E can be solved by various iterative gradient-based methods developed in non-linear optimization. In case of linear $F(x)$ this can be done analytically.

In some ways, nearest neighbor is at the opposite end of the spectrum from linear regression. It has little bias (assumptions about what the true function is), which allows it to fit non-linear functions without difficulty. Unfortunately, it suffers from high variance, which means noisy data causes it to make erratic predictions.

The distance weighted k-NN algorithm appears to be robust to noisy training data and effective when provided with sufficiently large set of training data. This is a local method, approximating the underlying target function locally, and this can help in smoothing out the impact of isolated noisy training examples. There is one problem, however, which is linked to the way the distance is calculated. In contrast to the decision tree method, for example, where instances are split on the basis of the most relevant attributes, the Euclidean distance involves all the attributes, independent of their relevancy to a particular classification problem. The distance between neighbors may be dominated by the large number of irrelevant attributes.

There are some possibilities to overcome this problem:

- to exclude the non-relevant attributes from consideration at the data preparation stage by using a methodology such as principal component analysis;

- to weight each attribute differently when calculating the distance between two instances.

It is interesting to note that when the number of training examples is very large, k-nearest neighbor method approaches the Bayesian optimal classification.

Figure 3.3 shows the local nature of k-NN in a 2-dimensional scenario. Each cylinder contains the k nearest points to be used to predict a new instance.

Table 3.1 summarizes main differences between linear and k-nearest neighbors methods.

3.4 Principal component analysis

Principal component analysis (PCA) is a methodology to identify patterns in data, and express it in such a way that their similarities and differences are highlighted.

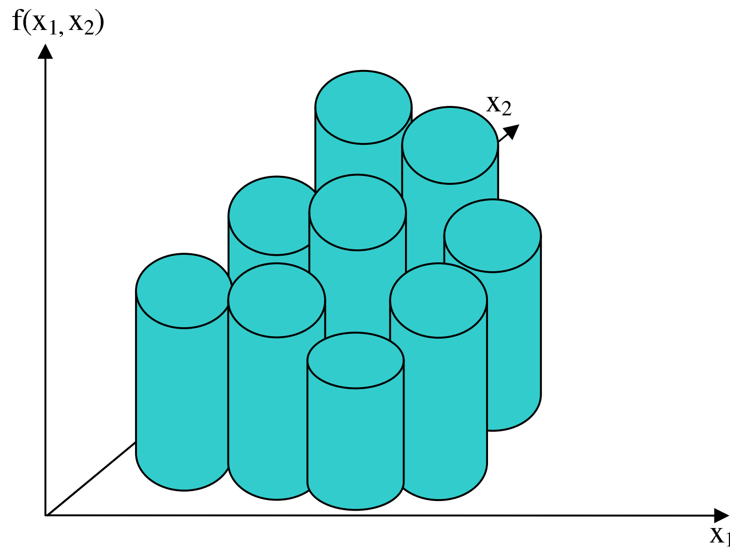

Figure 3.3. Locally constant nature of kNN regression.

Table 3.1. Main differences between linear and k-nearest neighbors methods.

Linear	**k-Nearest neighbors**
Builds a model first	Postpones building a model
Puts model in operation when prediction is taking place	Builds a model when new instance must be predicted
Uses all training data to build a model: globally linear	Uses just closest training set to build a model: locally constant or linear
Impact of noisy data is smoothed with the increase of training data	Noisy data can cause erratic predictions but it smooths impact of isolated noisy training data (local method)

Since patterns in data can be hard to find when talking about high dimensions, where graphical representation is not available, PCA is a powerful tool for analyzing data [75].

The other main advantage of PCA is that once one has found these patterns in the data, one can compress it, for example, by reducing the number of dimensions, without much loss of information. This technique is used, for example, in image compression.

Since PCA is a linear transformation with orthonormal basis vectors it can be expressed as a translation and rotation. Denoting the input data x and the transformed data y, the transformation can be expressed as

$$y = A(x - \mu_x) \tag{3.16}$$

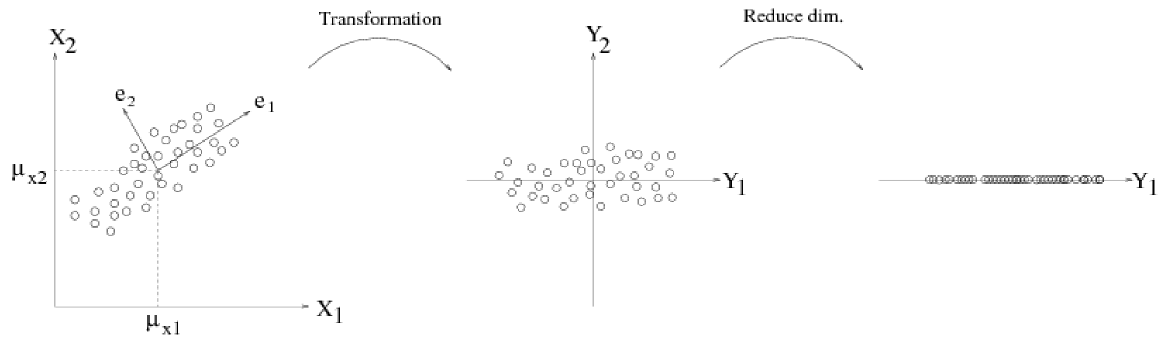

Figure 3.4. Basic principle of PCA in two dimensional case.

where A contains the new basis vectors, e_i as row vectors, hence $A = [e_1 e_2 ... e_n]^T$, and μ_x is the mean of the data set, hence $\mu_x = 1/n \sum_{i=1}^{n} x_i$.

In Figure 3.4 [75], the basic principles of PCA is illustrated for the two dimensional case.

The first figure in Figure 3.4 illustrates the input where the i'th sample is denoted $x_i = [x_{1i} x_{2i}]^T$. The second illustrates the transformed data where the i'th sample is denoted $y_i = [y_{1i} y_{2i}]^T$ and is calculated using the equation above.

The first figure in Figure 3.4 illustrates how the data are transformed into another representation where the main part of variance of the data is represented in the first variable, y_1. That is, if the second variable is ignored, as illustrated by second figure in Figure 3.4, the main variance of the data is kept. In many cases variance equals information, hence a more compact representation of the data/information is obtained.

The theories presented will be used in the generation of the statistics model to predict faster heuristics. First of all principal component analysis will be applied to eliminate correlated variables. Then multivariate regression will be applied to generate the engine estimators. At last, k-NN will be applied to decide upon which set of estimators to use for each validation data.

Chapter 4

Mean Variance Portfolio Theory

Almost everyone own a portfolio (group) of assets. This portfolio is likely to contain real assets such as a car, a house or a refrigerator, as well as financial assets such as stocks and bonds. The composition of the portfolio may be the result of a series of haphazard and unrelated decisions or it may be the result of deliberate planning.

An investor is faced with a choice from among an enormous number of assets. When one considers the number of possible assets and the various possible proportions in which each can be held, the decision process seems overwhelming.

All decision problems have certain elements in common. Any problem involves the delineation of alternatives, the selection of criteria for choosing among those alternatives, and, finally, the solution of the problem.

This chapter presents Mean Variance Portfolio theory [29] to help delineating a solution to the portfolio selection problem which can be applied to the formal verification context.

4.1 The Opportunity Set

Consider an investor who will receive with certainty an income of $10,000 in each of two years. Assume that the only investment available is a savings account yielding 5% per year. In addition, the investor can borrow money at a 5% rate. How much should the investor save and how much should he or she consume each year? The economic theory of choice proposes to solve this problem by splitting the analysis into two parts. First, specify those options that are available to the investor. Second, specify how to choose among these options.

The first part of the analysis is to determine the options open to the investor. One option available is to save nothing and consume $10,000 in each period. This

option is indicated by the point B in Figure 4.1. Another option would be to save all income in the first period and consume everything in the second. In the second period his savings account would be worth the $10,000 he saves in period 1 plus interest of 5% on the $10,000 or $10,500. Adding this to the second period income of $10,000 gives him a consumption in period 2 of $10,500 + $10,000 = $20,500. This is indicated by point A in Figure 4.1.

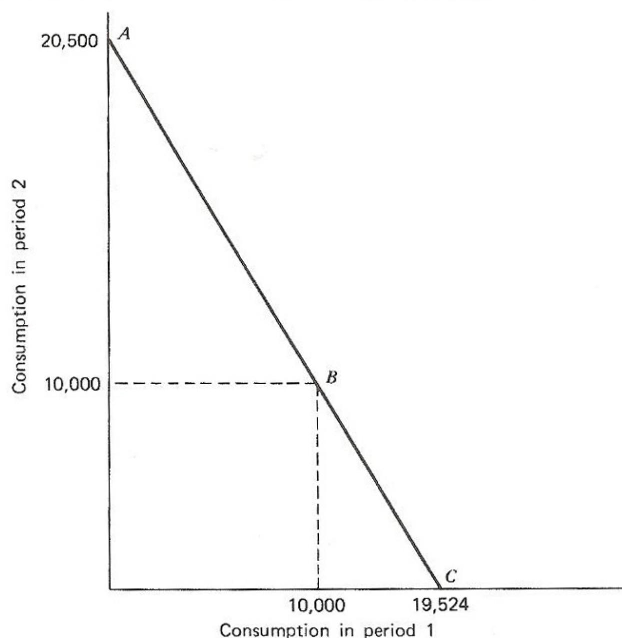

Figure 4.1. The investor's opportunity set.

Another possibility is to consume everything now and not worry about tomorrow. This would result in consumption of $10,000 from this period's income plus the maximum the investor could borrow against next period's income. If X is the amount borrowed, then X plus the interest paid for borrowing X equals the amount paid back. Since the investor's income the next period is $10,000, the maximum amount is borrowed if X plus the interest on X at 5% equals $10,000. Thus the maximum the investor can consume in the first period if $19,524. This is indicated by point C in Figure 4.1.

Note that points A, B and C lie along a straight line. In fact, all of the enormous possible patterns of consumption in periods 1 and 2 will lie along this straight line: $C_2 = \$20,500 - (1.05)C_1$. This is, of course, the equation for a straight line and is the line shown in Figure 4.1.

It has an intercept of $20,500, which results from zero consumption in period 1 ($C_1 = 0$) and is the point A determined earlier. It has a slope equal to -1.05 or minus

the quantity one plus the interest rate. The value of the slope reflects the fact that each dollar the investor consumes in period 1 is a dollar he cannot invest and, hence, reduces period 2's consumption by one dollar plus the interest he could earn on the dollar or a total of \$1.05. Thus an increase in period 1's consumption of a dollar reduces period 2's consumption by \$1.05.

The investor is left with a large number of choices. The set of choices facing the investor is usually referred to as the opportunity set.

4.2 The Indifference Curves

The economic theory of choice states that an investor chooses among the opportunities set shown in Figure 4.1 by specifying a series of curves called utility functions or indifference curves. A representative set is shown in Figure 4.2.

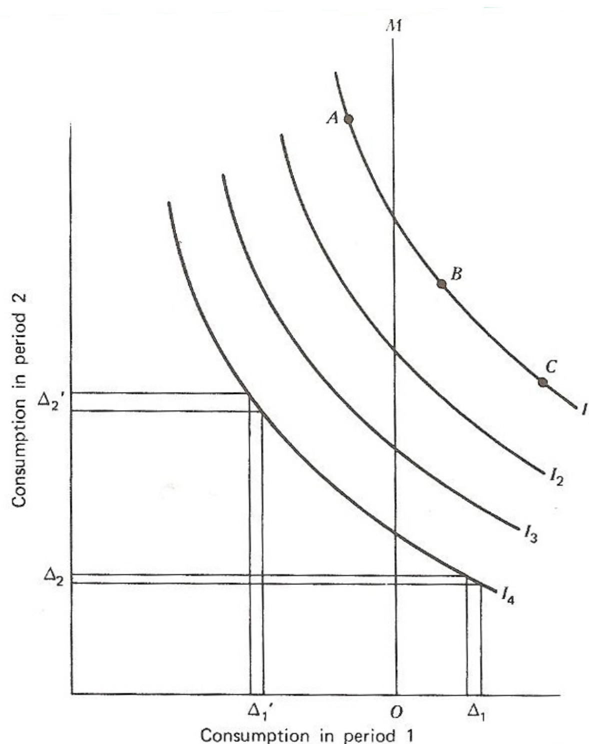

Figure 4.2. Indifference curves.

These curves represent the investor's preference for income in the two periods. The name "indifference curves" is used because the curves are constructed so that everyone along the same curve is assumed to be equally happy. In other words, the investor does not care whether he obtains point A, B or C along curve I_1.

Choices along I_1 will be preferred to choices along I_2, and choices on I_2 will be preferred to choices on I_3, and so on. This ordering results from an assumption that the investor prefers more than less. Consider the line OM. Along this line the amount of consumption in period 1 is held constant. As can be seen from Figure 4.2, along the line representing equal consumption in period 1, I_1 represents the most consumption in period 2, I_2 the next most, and so on. Thus, if the investors prefer more to less, I_1 dominates I_2, which dominates I_3.

The curved shape results from an assumption that each additional dollar of consumption forgone in period 1 requires greater consumption in 2. For example, if consumption in period 1 is large relative to consumption in period 2, the investor should be willing to give up a dollar of consumption in period 1 in return for a small increase in consumption in period 2. In Figure 4.2, this is illustrated by Δ_1 for the amount the investor gives up in period 1 and Δ_2 for the amount the investor gains in period 2. However, if the investor has very few dollars of consumption in period 1, then a large increase in 2 is required in order to be indifferent about giving up extra consumption in period 1. This is represented by Δ_1' in period 1 (which is the same size as Δ_1) and the Δ_2' in period 2, which is much larger than Δ_2.

The indifference curves and the opportunity set represent the tools necessary for the investor to reach a solution. The optimum consumption pattern for the investor is determined by the point at which a number of the set of indifference curves is tangent to the opportunity set (point D in Figure 4.3).

The investor can select either of the two consumption patterns indicated by the points where I_3 intersects the line ABC in Figure 4.3. But as the investor is better off selecting a consumption pattern lying on an indifference curve located above and to the right of I_3 if possible, he will move to higher indifference curves until the highest one that contains feasible consumption pattern is reached. That is the one tangent to the opportunity set. This is I_2 in Figure 4.3, and the consumption pattern the investor will choose is given by the point of tangency D.

If everyone knew with certainty the returns on all assets, then the framework just presented could easily be extended to multiple assets. If a second asset existed that yielded 10%, then opportunity set involving investment in this asset would be the line A'BC' shown in Figure 4.4.

Its intercept on the vertical axis would be \$10,000 + (1.10)(\$10,000) = \$21,000 and the slope would be -(1.10). If such an asset existed, the investor would surely prefer it if lending and prefer the 5% asset if borrowing. The preferred opportunity set would be A', B, C. Additional assets could be added in a straightforward manner.

But this situation is inherently unstable. Two assets yielding different certain re-

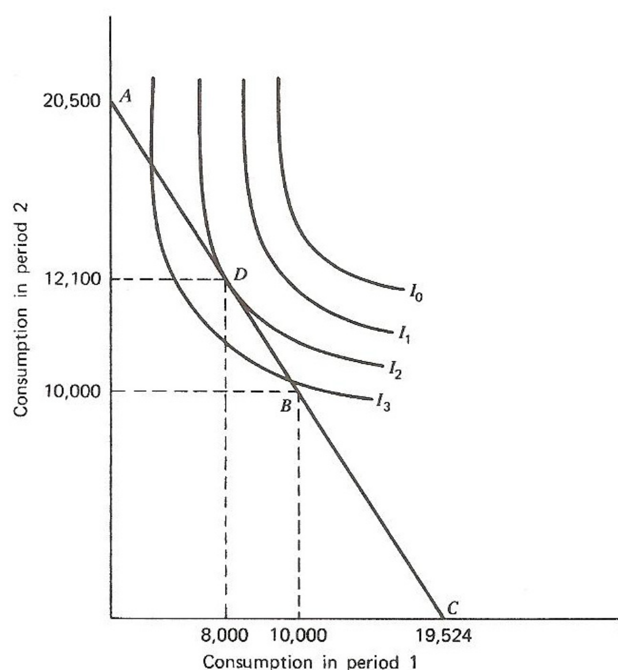

Figure 4.3. Investor equilibrium.

turns cannot both be available since everyone will want to invest in the higher yielding one and no one will purchase the lower yielding one. We are left with two possibilities: either there is only one interest rate available in the marketplace or returns are not certain (transaction costs or alternative tax treatment of income from different securities can explain the existence of some differential rates, but nothing like the variety and magnitude of differential found in the marketplace). Since we observe many different interest rates, uncertainty must play an important role in the determination of market rates of return. To deal with uncertainty, we need to develop a more complex opportunity set.

4.3 Opportunity Set Under Risk

Investors like high return, but don't like high risk.

We use return to indicate the return in an investment over a particular span of time called holding period return. Return will be measured by the sum of the change in the market price of a security plus any income received over a holding period divided by the price of a security at the beginning of the holding period. Thus, if a stock started the year at $100, paid $5 in dividends at the end of the year, and had a price of $105 at the end of the year, the return would be 10%.

Factors that affect securities risk include:

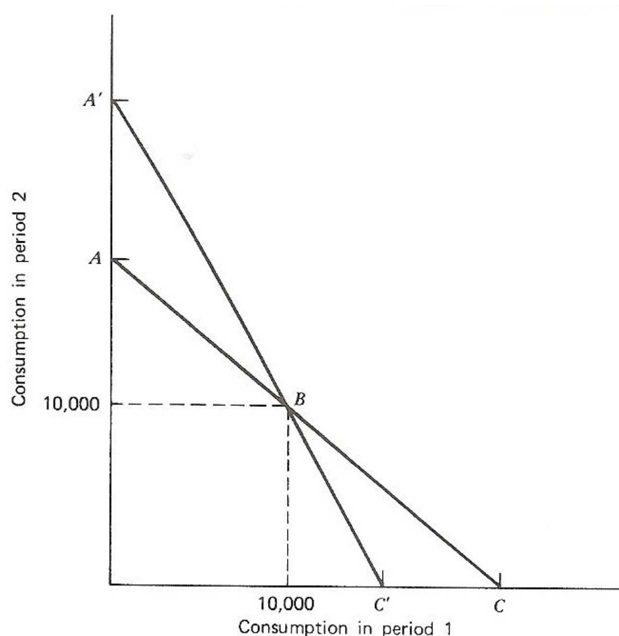

Figure 4.4. Investor's opportunity set with several alternatives.

- The maturity of an instrument (in general the longer the maturity the more risky it is).
- The risk characteristic and creditworthiness of the issuer or guarantor of the investment.
- The nature and priority of the claims the investment has on income and assets.
- The liquidity of the instrument and the type of market in which it is traded.

If risk is related to these elements, then measures of risk such as the variability of returns should be related to these same factors. A widely accepted measure of risk is the standard deviation of the returns along the time.

4.3.1 Return Distribution

The existence of risk means that the investor can no longer associate a single number or payoff with investment in any asset. The payoff must be described by a set of outcomes and each of their associated probability of occurrence, called a frequency function or return distribution.

A frequency function is a listing of all possible outcomes along with the probability of the occurrence of each. Table 4.1 shows such a function. This investment has three possible returns. If event 1 occurs, the investor receives a return of 12%; if event 2

Table 4.1. Frequency function.

Return	Probability	Event
12	1/3	1
9	1/3	2
6	1/3	3

occurs, 9% is received; and if event 3 occurs, 6% is received. In this example each of these events assumes to be equally likely.

Usually we do not delineate all of the possibilities as we have in Table 4.1. The possibilities for real assets are sufficiently numerous that developing a table like this for each asset is too complex a task. Furthermore, even if the investor decided to develop such tables, the inaccuracies introduced would be so large that he or she would probably be better off just trying to represent the possible outcomes in terms of some summary measures. In general, it takes at least two measures to capture the relevant information about frequency function:

1. a measure of central tendency or average value, called the expected return and
2. a measure of risk or dispersion around the mean, called the standard deviation or variance.

If R_{ij} denote the jth possible outcome for the return on security i and P_{ij} is the probability of the jth return on ith asset, then the average or expected return is:

$$\overline{R}_i = \sum_{j=1}^{M}(P_{ij}R_{ij}) \tag{4.1}$$

4.3.2 Variance

As we mentioned, it is also useful to have some measure of how much the outcomes differ from the average. The variance, or the average squared deviation, has some convenient properties and are usually used.

The formula for the variance of the return on the ith asset, α_i^2, when each return is equally likely is:

$$\alpha_i^2 = \sum_{j=1}^{M} \frac{(R_{ij} - \overline{R}_i)^2}{M} \tag{4.2}$$

If the observations are not equally likely, then, as before, we multiply the probability with which they occur:

$$\alpha_i^2 = \sum_{j=1}^{M} P_{ij}(R_{ij} - \overline{R}_i)^2 \tag{4.3}$$

4.4 Combination of Assets

The simple analysis of asset's mean and variance has taken us partway toward an understanding of the choice between risky assets. However, the options open to an investor are not to simply pick between assets, but also to consider combinations of these assets. For example, an investor could invest part of his money in each asset.

While this opportunity vastly increases the number of options open to the investor and hence the complexity of the problem, it also provides the "raison d'être" of portfolio theory.

The risk of a combination of assets is very different from a simple average of the risk of individual assets. Characteristics of the return on portfolios of assets can differ from the characteristics of the return on individual assets:

- When assets have their good and bad outcomes at different times, then investment in these assets can radically reduce the dispersion obtained by investing in one of the assets by itself. It the good outcomes of an asset are not always associated with the bad outcomes of a second asset, but the general tendency is in this direction, then the reduction in dispersion still occurs. However, it is still often true that appropriately selected combinations of the two assets will have less risk than the least risky of the two assets.

- When the conditions leading to various returns are different for the two assets (returns are independent), dispersion may be reduced, but not as drastic as before. With independent returns, extreme observations can still occur. They just occur less frequently. Just as the extreme values occur less frequently, outcomes closer to the mean become more likely so that the frequency function has less dispersion.

- When assets being combined have their outcomes affected in the same way by the same events, characteristics of the portfolio may be identical to the characteristic of individual assets. In less extreme cases, this is no longer true. Insofar as the good and bad returns on assets tend to occur at the same time, but not always exactly at the same time, the dispersion on the portfolio of assets is somewhat reduced relative to the dispersion on the individual assets.

Therefore, the first summary characteristic is the return on a portfolio of assets, which is simply a weighted average of the return on the individual assets. The weight applied to each return is the fraction of the portfolio invested in that asset. If R_{pj} is the jth return on the portfolio and X_i is the fraction of the investor's funds invested in the ith asset, then:

$$R_{P_j} = \sum_{i=1}^{N} X_i R_{ij} \tag{4.4}$$

The expected return is also a weighted average of the expected returns on the individual assets. Taking the expected value of the expression just given for the return on a portfolio yields:

$$\overline{R}_{P_j} = E(R_{P_j}) = E(\sum_{i=1}^{N} X_i Rij) \tag{4.5}$$

But, using one property of expected value that says that the expected value of the sum of two returns is equal to the sum of the expected value of each return $(E(R_{1j} + R_{2j}) = \overline{R}_1 + \overline{R}_2)$:

$$\overline{R}_P = \sum_{i=1}^{N} E(X_i R_{ij}) \tag{4.6}$$

Finally, the expected value of a constant times a return is a constant times the expected return:

$$\overline{R}_P = \sum_{i=1}^{N} X_i \overline{R}_i \tag{4.7}$$

The second summary characteristic was the variance. The variance on a portfolio is a little more difficult to determine than the expected return. For a two asset example, the variance of a portfolio P, designated by

$$\sigma_P^2 \tag{4.8}$$

is simply the expected value of the squared deviations of the return on the portfolio from the mean return on the portfolio, or

$$\sigma_P^2 = E(R_p - \overline{R}_p)^2 \tag{4.9}$$

Substituting in this expression the formulas for return on the portfolio and mean return yields in the two-security case:

$$\sigma_P^2 = E(R_p - \overline{R}_p)^2 = E[X_1R_{1j} + X_2R_{2j} - (X_1\overline{R}_1 + X_2\overline{R}_2)]^2 \tag{4.10}$$

$$\sigma_P^2 = E[X_1(R_{1j} - \overline{R}_1) + X_2(R_{2j} - \overline{R}_2)]^2 \tag{4.11}$$

where $\overline{R}_i$ stands for the expected value of security i with respect to all possible outcomes. Recall that

$$(X + Y)^2 = X^2 + XY + XY + Y^2 = X^2 + 2XY + Y^2 \tag{4.12}$$

Applying this to the previous expression we have:

$$\sigma_P^2 = E[X_1^2(R_{1j} - \overline{R}_1)^2 + 2X_1X_2(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2) + X_2^2(R_{2j} - \overline{R}_2)^2] \tag{4.13}$$

Applying the two rules that the expected value of the sum of a series of return is equal to the sum of the expected value of each return, and that the expected value of a constant times a return is equal to the constant times the expected return, we have:

$$\sigma_P^2 = X_1^2E[(R_{1j} - \overline{R}_1)^2] + 2X_1X_2E[(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2)] + X_2^2E[(R_{2j} - \overline{R}_2)^2] \tag{4.14}$$

$$\sigma_P^2 = X_1^2\sigma_1^2 + 2X_1X_2E[(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2)] + X_2^2\sigma_2^2 \tag{4.15}$$

$E[(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2)]$ has a special name. It is called the covariance and will be designated as σ_{12}. Substituting the symbol σ_{12} for $E[(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2)]$ yields:

$$\sigma_P^2 = X_1^2\sigma_1^2 + X_2^2\sigma_2^2 + 2X_1X_2\sigma_{12} \tag{4.16}$$

Notice what the covariance does. It is the expected value of the product of two deviations: the deviations of the returns on security 1 from its mean $(R_{1j} - \overline{R}_1)$ and the deviations of security 2 from its mean $(R_{2j} - \overline{R}_2)$. In this sense it is very much like the variance. However, it is the product of two different deviations. As such it can be positive or negative. It will be large when the good outcomes for each stock occur together and when the bad outcomes for each stock occur together. In this case, for good outcomes the covariance will be the product of two large positive numbers, which is positive. When bad outcomes occur, the covariance will be the product of two large negative numbers, which is positive. This will result in a large value for the covariance and a large variance for the portfolio. In contrast, if good outcomes for one asset are associated with bad outcomes of the other, the covariance is negative. It is

negative because a plus deviation for one asset is associated with a minus deviation for the second and the product of a plus and a minus is negative.

The covariance is a measure of how returns on assets move together. For many purposes, it is useful to standardize the covariance. Dividing the covariance between two assets by the product of the standard deviation of each asset produces a variable with the same properties as the covariance but with a range of -1 to +1. The measure is called the correlation coefficient. Letting ρ_{ik} stand for the correlation between securities i and k the correlation coefficient is defined as

$$\rho_{ik} = \frac{\sigma_{ik}}{\sigma_i \sigma_k} \tag{4.17}$$

The formula for variance of a portfolio, 4.16, can be generalized to more than two assets. Consider first a three-asset case. Substituting the expression for return on a portfolio and expected return of a portfolio in the general formula for variance yields

$\sigma_P^2 = E(R_P - \overline{R}_P)^2 = E[X_1 R_{1j} + X_2 R_{2j} + X_3 R_{3j} - (X_1 \overline{R}_1 + X_2 \overline{R}_2 + X_3 \overline{R}_3)]^2$

Rearranging,

$\sigma_P^2 = E[X_1(R_{1j} - \overline{R}_1) + X_2(R_{2j} - \overline{R}_2) + X_3(R_{3j} - \overline{R}_3)]^2$

Squaring the right-hand side yields

$\sigma_P^2 = X_1^2(R_{1j} - \overline{R}_1)^2 + X_2^2(R_{2j} - \overline{R}_2)^2 + X_3^2(R_{3j} - \overline{R}_3)^2 + 2X_1X_2E[(R_{1j} - \overline{R}_1)(R_{2j} - \overline{R}_2)] + 2X_1X_3E[(R_{1j} - \overline{R}_1)(R_{3j} - \overline{R}_3)] + 2X_2X_3E[(R_{2j} - \overline{R}_2)(R_{3j} - \overline{R}_3)]$

Utilizing σ_i^2 for variance of asset i and σ_{ij} for the covariance between assets i and j, we have

$\sigma_P^2 = X_1^2\sigma_1^2 + X_2^2\sigma_2^2 + X_3^2\sigma_3^2 + 2X_1X_2\sigma_{12} + 2X_1X_3\sigma_{13} + 2X_2X_3\sigma_{23}$

This formula can be extended to any number of assets. Examining the expression for the variance of a portfolio of three assets should indicate how. First note that the variance of each asset is multiplied by the square of the proportion invested in it. Thus, the first part of the expression for the variance of a portfolio is the sum of the variances on individual assets times the square of the proportion invested in each, or

$$\sum_{i=1}^{N} X_i^2 \sigma_i^2 \tag{4.18}$$

The second set of terms in the expression for the variance of a portfolio is covariance terms. Note that the covariance between each pair of assets in the portfolio enters the expression for the variance of a portfolio. With three assets the covariance between 1 and 2, 1 and 3, and 2 and 3 entered. With four assets, covariance terms 1 and 2, 1 and 3, 1 and 4, 2 and 3, 2 and 4, and 3 and 4 would enter. Further note that each covariance term is multiplied by two times the product of the proportions invested in

each asset. The following double summation captures the covariance terms:

$$\sum_{j=1}^{N} \sum_{[k=1,k!=j]}^{N} X_j X_k \sigma_{jk} \tag{4.19}$$

Putting together the variance and the covariance parts of the general expression for the variance of a portfolio yields:

$$\sigma_P^2 = \sum_{j=1}^{N} X_j^2 \sigma_j^2 + \sum_{j=1}^{N} \sum_{[k=1,k!=j]}^{N} X_j X_k \sigma_{jk} \tag{4.20}$$

4.5 Efficient Frontier

In theory we could plot all conceivable risky assets and combinations of risky assets in a diagram in return standard deviation space. We used the words "in theory", not because there is a problem in calculating the risk and return on a stock or portfolio, but because there are an infinite number of possibilities that must be considered. Not only must all possible groupings of risky assets be considered, but all groupings must be considered in all possible percentage compositions.

If we were to plot all possibilities in risk-return space, we would get a diagram like Figure 4.5. We have taken the liberty of representing combinations as a finite number of points in constructing the diagram. Let us examine the diagram and see if we can eliminate any part of it from consideration by the investor. As already reasoned, an investor would prefer more return to less and would prefer less risk to more. Thus, if we could find a set of portfolios that

1. offered a bigger return for the same risk, or
2. offered a lower risk for the same return,

we would have identified all portfolios an investor could consider holding. All other portfolios could be ignored.

Take a look at Figure 4.5. Examine portfolios A and B. Note that portfolio B would be preferred by all investors to portfolio A because it offers a higher return with the same level of risk. We can also see that portfolio C would be preferable to portfolio A because it offers less risk at the same level of return. Notice that at this point in our analysis we can find no portfolio that dominates portfolio C or portfolio B. It should be obvious at this point that an efficient set of portfolios cannot include interior portfolios. We can reduce the possibility set even further. For any point in risk-return

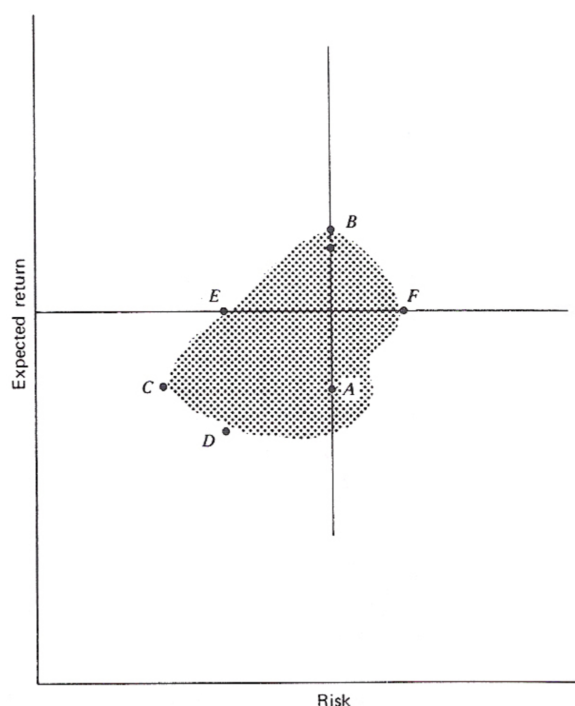

Figure 4.5. Risk and return possibilities for various assets and portfolios.

space we want to move as far as possible in the direction of increasing return and as far as possible in the direction of decreasing risk. Examine the point D, which is an exterior point. We can eliminate D from further consideration since portfolio E exists, which has more return for the same risk. This is true for for every other portfolio as we move up the outer shell from D to point C. Point C cannot be eliminated since there is no portfolio that has less risk for the same return or more return for the same risk. But what is point C? It is the global minimum variance portfolio. Now examine point F. Point F is on the outer shell, but point E has less risk for the same return. As we move up the outer shell curve from point F, all portfolios are dominated until we come to portfolio B. Portfolio B cannot be eliminated for there is no portfolio that has the same return and less risk or the same risk and more return than point B. Point B represents that portfolio (usually a single security) that offers the highest expected return of all portfolios. Thus the efficient set consists of the envelope curve of all portfolios that lie between the global minimum variance portfolio and the maximum return portfolio. This set of portfolios is called the "efficient frontier".

Figure 4.6 represents a graph of the efficient frontier. Notice that we have drawn the efficient frontier as a concave function. The proof that it must be concave follows logically from the earlier analysis of the combination of two securities or portfolios.

The portfolio problem, then, is to find all portfolios along this frontier.

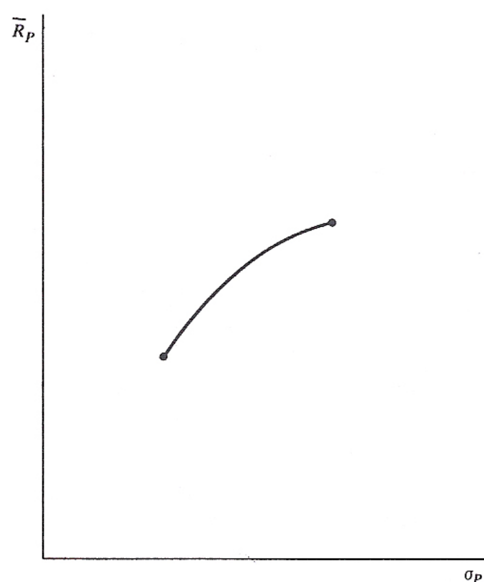

Figure 4.6. The efficient frontier.

In the scenario where there is a riskless lending and borrowing rate, there is a single portfolio of risky assets that is preferred to all other portfolios. Furthermore, in return standard deviation space, this portfolio plots on the ray connecting the riskless asset and a risk portfolio that lies furthest in the counterclockwise direction. For example, in Figure 4.7, the portfolio on the ray $R_F - B$ is preferred to all other portfolios of risky assets. The efficient frontier is the entire length of the ray extending through R_F and B. Different points along the ray $R_F - B$ represent different amounts of borrowing and/or lending in combination with the optimum portfolio of risky assets portfolio B.

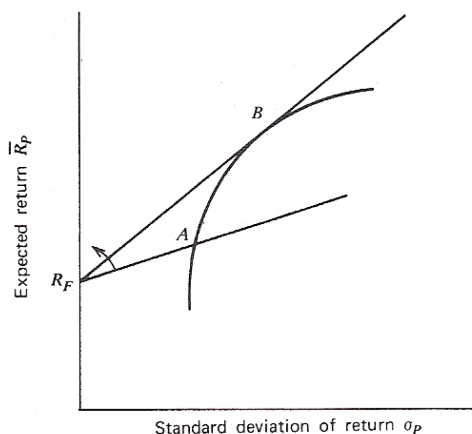

Figure 4.7. Combinations of the riskless asset in a risky portfolio.

An equivalent way of identifying the ray $R_F - B$ is to recognize that it is the ray

with the greatest slope. The efficient set is determined by finding that portfolio with the greatest ratio of excess return (expected return minus risk-free rate) to standard deviation that satisfies the constraint that the sum of the proportions invested in the assets equals 1. In equation form we have: maximize the objective function

$$\theta = \frac{\overline{R}_P - R_F}{\sigma_P} \tag{4.21}$$

subject to the constraint

$$\sum_{i=1}^{N} X_i = 1 \tag{4.22}$$

This is a constrained maximization problem. There are standard solution techniques available for solving it. For example, it can be solved by the method of Lagrangian multipliers. There is an alternative. The constraint could be substituted into the objective function maximized as in an unconstrained problem. This latter procedure will be followed below. We can write R_F as R_F times 1. Thus we have

$$R_F = 1R_F = (\sum_{i=1}^{N} X_i)R_F = \sum_{i=1}^{N} X_i R_F \tag{4.23}$$

Making this substitution in the objective function and stating the expected return (4.19) and standard deviation of return (4.20) in general form, yields

$$\theta = \frac{\sum_{i=1}^{N} X_i(\overline{R}_i - R_F)}{[\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{[j=1,j!=i]}^{N} X_i X_j \sigma_{ij}]^{1/2}} \tag{4.24}$$

This maximization problem is a very simple maximization problem and as such can be solved using the standard methods of basic calculus. In calculus it is shown that to find the maximum of a function you take the derivative with respect to each variable and set it equal to zero[1]. Thus the solution to the maximization problem just presented involves finding the solution to the following system of simultaneous equations:

1. $\frac{d\theta}{dX_1} = 0$

[1]Solving the problem without constraining the solution by

$$\sum_{i=1}^{N} X_i = 1 \tag{4.25}$$

does not work in every maximization problem. It works here because the equations are homogeneous of degree zero.

2. $\frac{d\theta}{dX_2} = 0$
3. $\frac{d\theta}{dX_3} = 0$

$\vdots$

N. $\frac{d\theta}{dX_N} = 0$

Two rules from the calculus are needed:

1. **The product rule**: θ is the product of two functions. The product rule states that the derivative of the product of two functions is the first function times the derivative of the second function plus the second times the derivative of the first. In symbols,

$$\frac{d}{dx}[[F_1(x)][F_2(x)]] = [F_1(x)]\frac{dF_2(x)}{dx} + [F_2(X)]\frac{dF_1(x)}{dx} \tag{4.26}$$

Let

$$F_1(X) = \sum_{i=1}^{N} X_i(\overline{R_i} - R_F) \tag{4.27}$$

$$F_2(X) = (\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij}) \tag{4.28}$$

Consider the derivative of $F_1(X)$. The derivative of the terms not involving X_k are zero (they are constants as far as X_k is concerned). The derivative of the term involving X_k is $\overline{R_k} - R_F$. Thus

$$\frac{dF_1(X)}{dX_k} = \overline{R_k} - R_F \tag{4.29}$$

Now consider the derivative of $F_2(X)$. To determine, the second rule from calculus is needed.

2. **The chain rule**: $F_2(X)$ involves a term in brackets to a power (the power $-\frac{1}{2}$). The chain rule states that its derivative is the power, times the expression in parentheses to the power minus one, times the derivative of what is inside the brackets. Thus,

$$\frac{dF_2(X)}{dX_k} = (-\frac{1}{2})(\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij})^{-\frac{1}{2}} \times (2X_k\sigma_k^2 + 2\sum_{j=1,j!=k}^{N} X_j\sigma_{jk}) \tag{4.30}$$

The only term that requires comment is the last one. The derivative of

$$\sum_{i=1}^{N} X_i^2\sigma_i^2 \tag{4.31}$$

follows the same principle discussed earlier. All terms not involving k are constant as far as k is concerned and thus their derivative is zero. The term involving k is $X_k^2\sigma_k^2$ and has a derivative of $2X_k\sigma_k^2$. The derivative of the double summation is more complex. Consider the double summation term

$$(\sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij}) \tag{4.32}$$

We get X_k twice, once when $i = k$ and once when $j = k$. When $i = k$, we have

$$\sum_{j=1,j!=ki}^{N} X_kX_j\sigma_{kj} = X_k[\sum_{j=1,j!=k}^{N} X_j\sigma_{kj}] \tag{4.33}$$

The derivative of this is, of course

$$\sum_{j=1,j!=k}^{N} X_j\sigma_{kj} \tag{4.34}$$

Similarly, when $j = k$, we have

$$\sum_{i=1,i!=ki}^{N} X_iX_k\sigma_{ik} = X_k[\sum_{i=1,i!=k}^{N} X_i\sigma_{ik}] \tag{4.35}$$

The derivative of this is also

$$\sum_{i=1,i!=k}^{N} X_i\sigma_{ik} \tag{4.36}$$

i and j are simply summonds. It does not matter which we use. Further, $\sigma_{ik} = \sigma_{ki}$. Thus

$$\sum_{j=1,j!=k}^{N} X_j\sigma_{kj} = \sum_{i=1,i!=k}^{N} X_i\sigma_{ik} \tag{4.37}$$

and we have the expression shown in the derivative, namely,

$$2\sum_{j=1,j!=k}^{N} X_j\sigma_{kj} \tag{4.38}$$

Substituting (4.27), (4.28), (4.29) and (4.30) into the product rule, expression (4.26) yields

$$\begin{aligned}\frac{d\theta}{dX_k} &= [\sum_{i=1}^{N} X_i(\overline{R_i} - R_F)][(-\frac{1}{2})(\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij})]^{-\frac{3}{2}}\\ &\times [(2X_k\sigma_k^2 + 2\sum_{j=1,j!=k}^{N} X_j\sigma_{kj})] + (\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij}]^{-\frac{1}{2}}\\ &\times [(\overline{R_k} - R_F)] = 0\end{aligned}$$

Multiplying the derivative by

$$(\textstyle\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij})^{\frac{1}{2}}$$

and rearranging yields

$$-[\frac{\sum_{i=1}^{N} X_i(\overline{R_i} - R_F)}{\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij}}][X_k\sigma_k^2 + \sum_{j=1,j!=k}^{N} X_j\sigma_{kj}] + (\overline{R_k} - R_F) = 0$$

Defining λ as

$$\frac{\sum_{i=1}^{N} X_i(\overline{R_i} - R_F)}{\sum_{i=1}^{N} X_i^2\sigma_i^2 + \sum_{i=1}^{N}\sum_{j=1,j!=i}^{N} X_iX_j\sigma_{ij}}$$

yields

$$-\lambda[X_k\sigma_k^2 + \sum_{j=1,j!=k}^{N} X_j\sigma_{kj}] + (\overline{R_k} - R_F) = 0$$

Multiplying the terms in the brackets by λ yields

$$-[\lambda X_k\sigma_k^2 + \sum_{j=1,j!=k}^{N} \lambda X_j\sigma_{kj}] + (\overline{R_k} - R_F) = 0$$

A mathematical trick allows a useful modification of the derivative. Note that each X_k is multiplied by a constant λ. Define a new variable $Z_k = \lambda X_k$. The X_k are the fraction to invest in each security, an the Z_k are proportional to this fraction. Substituting Z_k for the λX_k simplifies the formulation. To solve for the X_k after obtaining the Z_k, one divides each Z_k by the sum of the Z_k. Substituting Z_k for λX_k and moving the variance covariance terms to the right-hand side of the equality yields

$$\overline{R_i} - R_F = Z_1\sigma_{1i} + Z_2\sigma_{2i} + ... + Z_i\sigma_i^2 + ... + Z_{N-1}\sigma_{N-1i} + Z_N\sigma_{Ni}$$

We have one equation like this for each value of i. Thus the solution involves solving a system of simultaneous equations for the Z‘s.

The Z‘s are proportional to the optimum amount to invest in each security. There are N equations (one for each security) and N unknowns (the Z_k for each security). Then the optimum proportions to invest in stock K is X_k, where

$$X_k = \frac{Z_k}{\sum_{i=1}^{N} Z_i} \tag{4.39}$$

The Mean Variance Portfolio Theory will be applied in the proposed solution. Selecting the best n engines will base not just in performance maximization, taking the engines with smaller engine estimated times (maximum asset return), but it will also

base in the correlation among engines, taking engines with small correlation (minimum risk).

Chapter 5

A Multivariate Calibration Model to Predict Faster Verification Heuristics

As already mentioned, formal verification is an NP-complete problem in the size of the trace description (Binary Decision Diagrams state-explosion problem) and in the verification time (SAT solvers). Therefore, as many other NP-complete problems, practical solutions implement heuristics to try to solve efficiently such problems.

Each practical solution is known as engine (or solver). The engines usually are made up of different heuristics and algorithms, each of it having a set of configuration parameters. Since in functional verification, complexity is often measured by the size of the design space, which is exponential in the number of storage elements (flip-flops) in the design [78], engines performance is completely dependent on IC design and property to be verified, as this defines the number of storage elements in it. It becomes then necessary to identify and apply the best heuristics and algorithms to each IC property to be proved, what will allow speeding up of verification process and increasing the chance of reaching proof conclusion.

The performance of different heuristics rely on some design characteristics as well as on the property to be proved. Defining variables that express design/property behavior is part of this work.

The proposed methodology is comprised of the following steps:

1. Define independent variables ($\vec{x}$) that express design/property behavior;

2. Build a data set with independent variables metrics and engines execution times for each engine to be analyzed;

Table 5.1. Symbols convention.

Symbol	Meaning
$\vec{x}$	independent parameters
$\vec{p}$	dependent parameter
b	parameter coefficient
e_i	execution time of engine i
w_i	decision variable that selects engine i
$\tilde{}$	estimator
$\tilde{e}_i$	estimated execution time for engine i
σ	variance of sampled execution time
V	covariance matrix whose elements are σ_{ij}
θ	auxiliary function using variance σ

3. Data set partition into proof result groups: proven, falsified, inconclusive (time-out);
4. Apply principal component analysis and multivariate regression to each group's data set in order to generate engines' estimators;
5. Select best engines based on selection mechanisms:
   - Maximize performance (smaller execution times) and
   - Maximize performance and minimize correlation (greater chance of arriving to a proof conclusion, which is considered success).

5.1 Independent Variables ($\vec{x}$)

Choosing the set of independent variables $\vec{x}$ affects directly the quality of the estimators $\tilde{e}$, since they are supposed to explain the behavior of a given verification problem. However, since the problem is hard, we will always have a chance that the chosen variables won't be enough.

Independent variables have been chosen based on the literature, including structural and testability metrics.

1. Property circuit level

   Property circuit is the circuit in the cone of influence (COI) of a property. The COI is the set of system variables resulted from an abstraction technique that removes all variables from the system model that do not have any effect on the system properties.

The property circuit level (PCL) can be computed by

$$LG = 1 + \max LG_i \tag{5.1}$$

where LG_i is the level of each gate connected to the input ports of the gate being computed. Note that property circuit level computation needs to start from circuit inputs up to circuit outputs.

2. SCOAP Cycles

   Number of cycles to converge SCOAP [42] calculation. SCOAP is a testability measure that gives integral numerical estimates of the controllability and observability of signal lines in a given circuit. Appendix A.1 presents detailed information.

3. Reset state UNDEF

   Number of flops initialized with undefined state (x, z).

4. Reset state DEF

   Number of flops initialized with any defined state (0, 1).

5. Circuit TI (SCOAP testability index)

$$TI = \sum_{i=0}^{n} (CC0i + CC1i) \tag{5.2}$$

   where CC0 is combinational 0-controllability and CC1 is combinational 1-controllability. It guesses the capability of controlling each design flop.

6. Property TI

$$TI_{Prop} = \log(CC0_p + CC1_p) \tag{5.3}$$

   where p stands for property flop. It guesses the capability of controlling the property flop.

7. SCOAP Adjusted Flops

$$AdjFlops = \sum_{i=0}^{n} \frac{\log(CC0_i + CC1_i)}{\max(CC0_i + CC1_1)} \tag{5.4}$$

This metric gives the ability to control each design flop, related to the most difficult flop to control.

8. SCMax

   Greatest SCOAP [42] sequential controllability (SC) among all flops.

9. Flops

   Number of flops in cone of influence of the property.

10. Gate Bits

    Number of gate bits in the COI of the property.

11. Free Variables

    Sum of primary input and stopat bits in the COI of the property.

12. Constrained Bits

    Sum of flop bits of all connected assumes.

13. Counter Bits

    For each counter i in the design:

$$CounterBits = \frac{\ln \sum_{i=0}^{n} 2^{numBits_i}}{ln2} \tag{5.5}$$

where n is the number of counters in the design. This measure is an heuristic gives an intermediary metric number, between a pessimistic metric analysis ($2^{\sum_{i=0}^{n} numBits_i}$) and an optimistic analysis ($2^{max(numBits_i)}$), given that all counters may not be exercised at the same time.

14. Finite State Machine (FSM) Bits

    For each FSM i in the design:

$$FSMBits = \frac{\ln \sum_{i=0}^{n} 2^{numBits_i}}{ln2} \tag{5.6}$$

where n is the number of FSMs in the design. This measure is an heuristic that gives an intermediary metric number, between a pessimistic metric analysis ($2^{\sum_{i=0}^{n} numBits_i}$) and an optimistic analysis ($2^{max(numBits_i)}$), given that all FSMs may not be exercised at the same time.

15. Array Bits

    For each array i in the design:

$$ArrayBits = \frac{\ln \sum_{i=0}^{n} 2^{numBits_i}}{ln2} \tag{5.7}$$

    where n is the number of arrays in the design. This measure is an heuristic that gives an intermediary metric number, between a pessimistic metric analysis ($2^{\sum_{i=0}^{n} numBits_i}$) and an optimistic analysis ($2^{max(numBits_i)}$), given that all arrays may not be exercised at the same time.

5.2 Data set

Two thousand five hundred and thirty-three proof targets have been collected on different designs: not one design for all targets, neither one design for each target. Designs and properties were described in Verilog, VHDL, SystemVerilog and PSL. Proof times for 4 different engines have been collected: e_1, e_2, e_3 and e_4.

Besides collecting proof times and final results (proven true, counterexample found or proof timeout), for each proof target the, 15 structural and testability variables mentioned in section 5.1 were measured. It is important to note that we could not use normal formal verification benchmarks here as we required full access to the RTL designs and properties in order to compute the 15 independent variables metrics.

5.2.1 Data set partition

In order to get better results, the data set was classified in three main clusters, according to proof results:

- True

  Group that arrived to a valid proof result.

- CEX

  Group that found a counterexample as proof result.

- Timeout

  Group that arrived to an inconclusive proof result.

Estimators will be generated just for “True" and “CEX" groups, as “Timeout” represents incomplete proofs and a bounded proof methodology needs to be applied in such case (proposed as future work).

From the 2533 proof targets, 1024 were rejected due to small proof times: proof times smaller than 0.5 seconds for all engines were rejected since the objective of this work is to estimate engines for time consuming proofs.

From the 1509 remaining proof targets, 179 represent “CEX" group and 1330 represent “True" group.

5.3 Engine Estimators

The engines' selection methodologies (5.4.1 and 5.4.3) depend heavily on how well we can estimate the execution time, or how well we can estimate the relative performance of the engines, which is given by the estimated time $\tilde{e}$ of engines:

$$\tilde{e}_i(\vec{x}) \tag{5.8}$$

which returns the estimated time of engine e_i to prove a property whose design is characterized by $\vec{x}$, a vector of design metrics (the independent variables).

5.3.1 Linearization of Exponential Equations

As a consequence of the NP-completeness of the verification process, we consider $\tilde{e}_i$ to have some exponential nature on the size of the circuit, for some parameter j:

$$\tilde{e}_i \; \alpha \; {b_j}^{x_j} \tag{5.9}$$

where $\vec{x}$ is a set of variables, composed by metrics that characterizes the design and property to be proved.

Estimation time function $\tilde{e}$ of verification process can be described as following exponential function:

$$\tilde{e}(\vec{x}) = b_0 \prod_{i}^{|X|} b_i^{x_i} \tag{5.10}$$

where $\vec{x}$ contains the metrics that characterizes the design and property to be proved.

Linearization of $\tilde{e}_i$ is done directly by getting the logarithm of both sides of the formula:

$$\log \tilde{e}(\vec{x}) = \log b_0 + \sum_{i}^{|X|} x_i \log b_i \tag{5.11}$$

Such linearization allows the application of linear regression methods to get prediction models.

5.3.2 Polynomial Effect of Variables

In addition, we consider that engines are affected in different ways by $\vec{x}$ and some of the parameters may influence the estimated execution time $\tilde{e_i}$ in a polynomial manner, rather than exponential. As a result, for some of the parameters, we have for parameter k:

$$\tilde{e_i} \; \alpha \; {x_k}^{b_k} \tag{5.12}$$

It is possible, for example, to have a variable x_i that has an exponential effect on one engine, while having a polynomial effect on another engine. It is important to note here that, although one or more metrics may have a polynomial effect over verification estimation time, the global effect is still exponential due to the exponential nature of the problem.

Polynomial effect of variables may be detected by analyzing the prediction model error. If the prediction error decreases when fixing a variable effect as polynomial, then this variable has probably a polynomial effect over the engine.

The algorithm below describes the methodology to identify variables with polynomial effect. Figure 5.1 illustrates such algorithm.

```
flag_continue = true;
error = error with linear regression;
while (flag_continue == true) {
    best_i = -1;
  foreach i from 0 to N {
      make xli = log(xi)
   make regression with xli in place of xi, keeping all
            other variables the same
      if (error(xli) < error)
      then {
          best_i = i;
          error = error(xli);
```

```
    }
  }
  if (best_i != -1)
  then keep xi as xli
  else flag_continue = false;
}
```

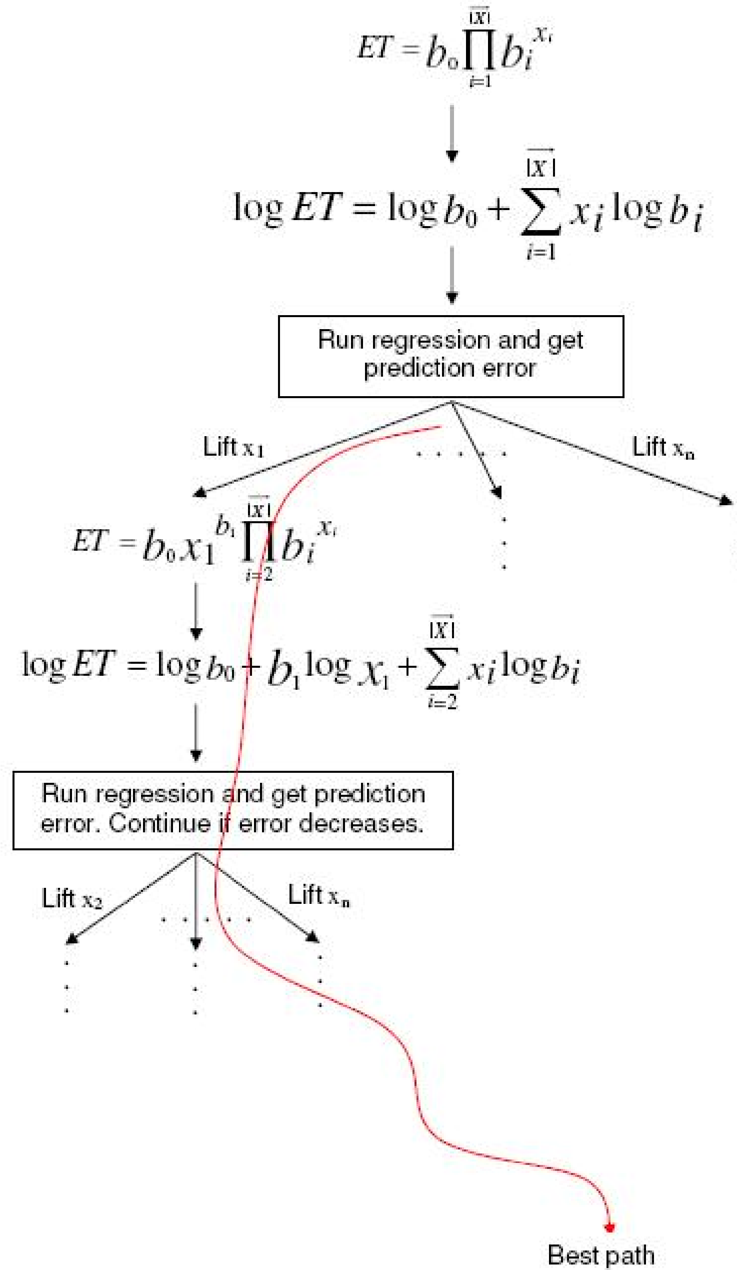

Figure 5.1. Methodology to identify variables with polynomial effect over an engine.

5.3.3 Multivariate regression

For the implementation of the proposed methodology, R tool was used. R is a language and environment for statistical computing and graphics. It is a GNU project which was developed at Bell Laboratories (formerly ATT, now Lucent Technologies). It includes:

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hard copy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

For analysis of the proposed methodology, an script was implemented in R with three phases: initialization, cross-validation and summarization. These three phases are presented in the following sections.

5.3.3.1 Initialization

In the initialization phase, data set was read into R, PCA was computed and an initial linear regression with all data has been done, as it can be seen in algorithm below.

```
Begin: initialization
    Read data: Cex (179 measures)
               Proven (1330 measures)
    Compute PCA: Cex group
                 Proven group
    Compute linear regression for engines e1, e2, e3 and e4:
        Cex group
        Proven group
    Get r squared for regression for each engine and each group
End: Initialization
```

In order to evaluate the quality of the estimators, Table 5.2 presents r-squared coefficient (squared Pearson's correlation coefficient [25]) for all estimators. This coefficient ranges from 0 to 1 and it indicates the correlation between the observed and

Table 5.2. r-squared for engine estimators with all data.

Engine	CEX Group	Proven Group
e_1	0.584	0.516
e_2	0.573	0.298
e_3	0.526	0.704
e_4	0.576	0.366

predicted execution times (0 indicates no relationship and 1 indicates a perfect linear relationship, with a value above 0.7 indicating a strong correlation).

It is important to recall that our main interest is to estimate the relative performance of the engines so that we can predict the best set of engines to run in parallel. In this sense, the estimators obtained were satisfactory, as they obtained reasonable relative performance with respect to the actual execution times.

5.3.3.2 Cross-validation

Cross-validation [53], sometimes called rotation estimation, is a technique for assessing how the results of an statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a sample of data into complementary subsets, performing the analysis on one subset (called the training set), and validating the analysis on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

Cross-validation has been applied to validate not just multivariate regression, but the whole methodology proposed. Training set counted for 67% of the data while validation set comprised the remaining 33%.

The partition and computation loop was executed one third of the size of the sample data (it means then 60 times for CEX group and 444 times for Proven group). It is important to remind that data set partition was done randomly.

Algorithm below presents the cross-validation loop for one group (it must be executed again for the other group). For each execution, all data computed was counted and summarized, in order to get, at the end, the average values and the percentages of each type of match.

```
Begin: Cross Validation
    Random sample data partition: training and validation sets.
```

```
    For each engine begin:
        Compute PCA for training set.
        Compute predict times for validation set,
            using training set PCA.
        Compute r-squared

    Compute covariance matrix for training set

    For each validation set entry:
              1. Compute smallest distance to entries
                   in training group
              2. Compute smallest distance to all entries
                   in other group
              3. Compute best engines based just in
                   execution time
              4. Compute best engines based in execution time
                   and correlation
End: Cross Validation
```

5.3.3.3 Summarization

In the summarization phase all computed data is prepared and final results are presented, as displayed in next section.

5.4 Engines' selection mechanisms

We mathematically formulated three forms of selection: "Maximize Performance" (select engines with smallest execution times) , "Minimize Correlation" (select engines with smallest correlation) and "Maximize Performance and Chance of Success" (select engines with smallest execution time and correlation).

In addition to selecting heuristics, the model can also be used to estimate and select the heuristics' parameters. Once we the best select engines, parameter values that define the selected engines are a good hint of possible good parameters.

5.4.1 Maximizing Performance

The first question a reader may ask is a very simple one - tell him/her which engine is the best engine to prove or to find a counterexample for a given property by maximizing the performance of the job (i.e. minimizing the execution time, $\tilde{e}$). Having $\tilde{e}_i$ as the estimated execution time for engine i, solution would consist in taking the smallest estimated execution time.

$$\min \sum_i w_i \tilde{e}_i \tag{5.13}$$

subject to

$$\sum_i w_i = 1$$
$$w_i \epsilon \ \{0, 1\}$$

Although we casted this problem as an optimization problem as it will help us understand our final solution, the solution to this problem can be easily obtained by just selecting the minimum estimate $\tilde{e}_i$.

5.4.2 Minimizing Correlation

It is interesting to to take into account during engines' selection the minimization of engines' correlation, as correlation is intimately related to our chance of success: by selecting engines that do not have similar behaviors we increase the chance of reaching a proof conclusion.

$$\min \sum_i w_i \sigma_i \tag{5.14}$$

subject to

$$\sum_i w_i = 1$$
$$w_i \epsilon \ \{0, 1\}$$

In this case, there is also a simple solution which is obtained by just selecting the engine i with minimum σ_i. We also casted this problem as an optimization problem to facilitate the understanding of the parallel selection problem.

5.4.3 Maximizing Performance and Chance of Success

Just like the Mean Portfolio Variance Theory (see Chapter 4), we could easily extend the Maximize Performance problem. In the Maximize Performance selection problem, the group of engines is selected even if their correlation is high. The selection of highly correlated heuristics could lead to similar behaviors and all selected engines could take, for example, a longer execution time than expected, arriving to no proof conclusion.

We can further improve this model and increase the chance of success by minimizing the selected engine's correlation to avoid similar behaviors.

$$\min \sum_i w_i \sigma_{ij} \tag{5.15}$$

subject to

$$\sum_i w_i = N \tag{5.16}$$

This can be rewritten as:

$$\min \sum_i \sum_j w_i w_j \sigma_{ij} \tag{5.17}$$

subject to

$$\sum_i w_i = N$$
$$w_i \epsilon \ \{0, 1\}$$

This last problem formulation is able to choose the engines that will lead to minimum correlation, even considering the dependency among them, but it does not consider performance (estimated time).

In order to address the problem of selecting multiple engines to increase the chance of success, we will consider the auxiliary function θ, adapted from the auxiliary function defined in Chapter 4, equation 5.18.

In the adaptation of the problem, the stock assets will be mapped to engine assets. So the portfolio will contain a set of engines to be chosen.

The auxiliary function θ proposed in 5.18 will be adapted to this environment:

$$\theta = \frac{\sum_{i=1}^{N} X_i(\overline{R}_i - R_F)}{[\sum_{i=1}^{N} X_i^2 \sigma_i^2 + \sum_{i=1}^{N} \sum_{[j=1,j!=i]}^{N} X_i X_j \sigma_{ij}]^{1/2}} \tag{5.18}$$

the numerator will be the performance estimate of the engine mix, and the denominator will be the correlation between the engines.

If we want to maximize performance, and performance is proportional to the inverse of $\tilde{e}$, we will use $\frac{1}{\tilde{e}}$ as the function to be maximized. So, substituting $(\overline{R}_i - R_F)$ by $\frac{1}{\tilde{e}_i}$, yields

$$\theta = \frac{\sum_i w_i(\frac{1}{\tilde{e}_i})}{[\sum_i {w_i}^2{\sigma_i}^2 + \sum_i \sum_j w_i w_j \sigma_{ij}]^{\frac{1}{2}}} \tag{5.19}$$

Like stated in Chapter 4, a good solution, thus, is obtained by maximizing the performance, and at the same time minimizing the correlation of the engines, thus yielding a maximization problem.

$$\max\ \theta \tag{5.20}$$

This problem can be solved by computing θ partial derivatives with respect to w_i and making them equal zero (see Section 4.5).

$$\frac{\partial \theta}{\partial w_i} = 0, \tag{5.21}$$

From 4.39 in Chapter 4:

$$\frac{\partial \theta}{\partial w_i} = -\lambda[w_i{\sigma_i}^2 + \sum_{j,j!=i} w_j\sigma_{ij}] + \frac{1}{\tilde{e}_i} = 0 \tag{5.22}$$

where

$$\lambda = \frac{\sum_i w_i(\frac{1}{\tilde{e}_i})}{[\sum_i w_i^2{\sigma_i}^2 + \sum_i \sum_{j,j!=i} w_i w_j \sigma_{ij}]} \tag{5.23}$$

As detailed in 4.5, the solution $\vec{w}$ is given by

$$\lambda \times V \times \vec{w} = (\frac{1}{\tilde{e}}) \tag{5.24}$$

The vector $\vec{w}$ specifies the weight we should give to each engine. As an approximation, for a n core machine, for example, we will take the n engines with highest weight in $\vec{w}$, maximizing the chance of return given by performance $\tilde{e}$, yet observing the minimum variance.

If you solve for all w_i, then λ is a constant. We are interested in ordering w, not on w absolute value itself. As we want to sort the w values to get the best engines, we can absorb their constant by w, as we are interested in the normalized w. By disregarding λ, we have:

$$V \times \vec{w} = (\frac{1}{\tilde{e}}) \tag{5.25}$$

Solving for $\vec{w}$ results in:

$$\vec{w} = V^{-1}(\frac{1}{\tilde{e}}) \tag{5.26}$$

Normalizing $\vec{w}$ as stated in 4.39 in Chapter 4::

$$\parallel \tilde{w} \parallel = \frac{\vec{w}}{\sum_i w_i} \tag{5.27}$$

Based on $\parallel \tilde{w} \parallel$, we can select the n highest elements of w_i to be the engines to execute in parallel.

Note that, although we showed how to obtain the best n engines, we did not consider the engine‘s parameters $\vec{p}$. This approach works by first estimating the values of $\vec{p}$, and then using the estimators $\tilde{e}$ based on the values obtained for $\vec{p}$. So the parameters are already defined for each engine.

It is important to observe that sometimes we can consider enumerating some of the dependent parameters p_j by considering an engine a solver with pre-defined parameters, and use all possible solver-parameter pairs as engines e_i when we find the maximum for θ like before.

For example, suppose we are selecting an heuristic for the SAT solver and that there are three options for some p_j.

$$Dom(p_j) = 0, 1, 2 \tag{5.28}$$

We would define the engines as $e_1(x, p = 1)$, $e_2(x, p = 2)$, $e_3(x, p = 3)$.

If we have many different parameters, finding engines with good performance may be a hint of good parameters selection. User can then explore these parameters configuration with small changes to even enhance performance.

Chapter 6

Results

In a scenario where we have two cores to prove a given target, this analysis consists in suggesting the two engines with the greater chance to solve the problem.

The data set is made up of collected proof times for 4 different engines: e_1, e_2, e_3, e_4. Data was then clustered into the following groups: “True" and “CEX". Estimators were generated for each cluster: e_{1True}, e_{1CEX}, e_{2True}, e_{2CEX}, e_{3True}, e_{3CEX}, e_{4True}, e_{4CEX}.

6.1 Multivariate regression

Linear regression was applied to data, according to Sections 5.3.2 (Polynomial Effect of Variables) and 5.3.1 (Linearization of Exponential Equations). Further analysis of the estimators enabled us to understand that the polynomial coefficients were providing good coverage for small values of the parameters, while the exponential coefficients in the estimators were providing good coverage when parameters values increased (larger values), as it can be seen in figure 6.1.

As a result, we ended up using the following model:

$$\tilde{e}(\vec{x}) = b_0 (\prod_{i}^{|X|} b_i^{x_i})(\prod_{j}^{|X|} x_j^{b_j}) \tag{6.1}$$

Linearization of equation above by getting the logarithm of both sides of the formula reaches:

$$\log \tilde{e}(\vec{x}) = \log b_0 + \sum_{i}^{|X|} x_i \log b_i + \sum_{j}^{|X|} b_j \log x_j \tag{6.2}$$

PCA, as discussed in Section 3.4, was applied before regression.

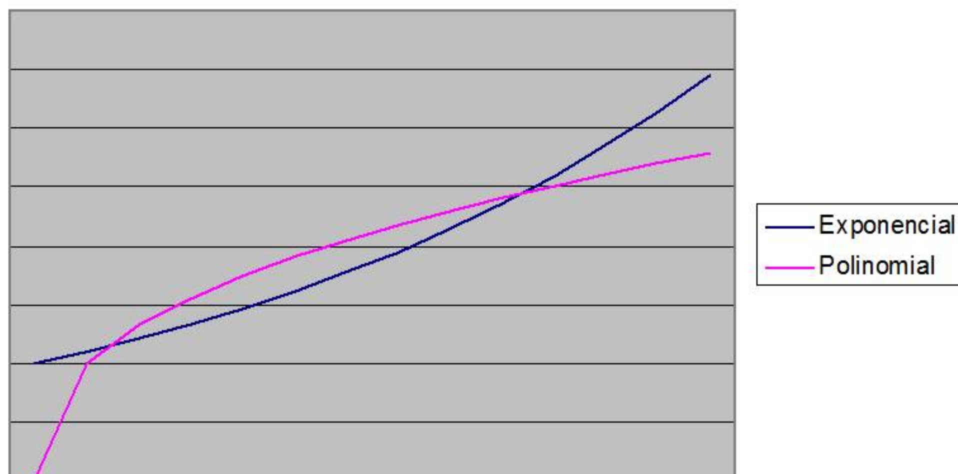

Figure 6.1. Polinomial effect for smaller values versus exponential effect for larger values.

Example of final estimator for e_{1CEX} is:

$$e_{1CEX} = b_0 + b_1 x_1 + ... + b_{15} x_{15} + b_{16} \log x_1 + ... + b_{30} \log x_{15} \tag{6.3}$$

6.2 Smallest distances

Users can employ the distance table to gain a feeling of which is the expected proof result in order to choose the specific estimators group ("True" or "CEX").

We computed the distance of the validation set to all entries in the the training set and also to all data in the other group. The formula used to compute such distance took into account x_k and lnx_k since best estimators were obtained when linear and logarithmic effects were considered.

$$d_i = min_k[\sum(x_i - x_k)^2 + \sum(lnx_i - lnx_k)^2]^{\frac{1}{2}} \tag{6.4}$$

Distance to each group's center of mass could not be used due to the big variability of the training set.

Table 6.1 presents the percentage of times an entry from the CEX validation set had smaller distance to an entry of CEX training set, and percentage of times and entry from CEX validation set had smaller distance to an entry of Proven group data. The same is presented for Proven group.

Since a user should start considering the use of "True" or "CEX" estimators to guide selection of the n engines, Table 6.1 brought great help. As it can be seen, for

Table 6.1. Percentage of times smaller distance matched correctly for each group at the end of validation process.

Group	Smallest distance found in CEX	Smallest distance found in Proven
CEX	60.75%	39.25%
Proven	6.02%	93.98%

CEX group, validation set entries were closer to an entry in CEX training set 60.75% of the times while for Proven group, validation set entries were closer to an entry in Proven training set 93.98% of the times. The size difference between CEX data set and Proven data set may be the reason for the worse result of CEX group compared to Proven group result.

6.3 Selection based on engine's time

During cross-validation, the following kinds of matches have been computed (suppose best engine is A and second best engine is B):

- Exact Match: number of times two best engines have been predicted correctly (A and B)
- 2 matches: number of times best engine was predicted as second best and second best was predicted as the best (B and A)
- 1 match Best: number of times best engine was predicted as best or second best, but second best engine was not predicted at all (A and X or X and A)
- 1 match: number of times second best engine was predicted as best or second best, but best engine was not predicted at all (B and X or X and B)
- No match: number of times neither the best nor the second best engines were predicted at all (X and Y)

Table 6.2 presents the results after cross-validation. As it can be seen, the best engine were selected as the first or second better 82.25% for CEX group and 90.91% for Proven group. In yellow we present the unsatisfactory results, which comprises the selections that do not include the best engine.

Table 6.2. Percentage of time of each kind of match for selection based in execution time.

Kind of match	CEX	Proven
Exact match	49.44%	45.49%
2 matches	19.00%	35.43%
1 match best	13.81%	9.99%
1 match	17.75%	9.09%
No match	0%	0%

6.4 Selection based on engine's time and correlation

In order to reduce risks of choosing the wrong engines to prove the property, the best solution was chosen by analyzing also the engine's correlation. A smaller engine correlation corresponds to better quality of solution (different behaviors) which, in turn, corresponds to the biggest chance of solving the problem.

The cluster's data set ("True" and "CEX") were analyzed separately, since we need to compute the covariance matrix of each group.

As stated in Section 5.4.3, the goal is to generate normalized vector $\parallel \tilde{w} \parallel$ as stated by Equation 5.27:

$$\parallel \tilde{w} \parallel = \frac{\vec{w}}{\sum_i w_i} \tag{6.5}$$

$\parallel \tilde{w} \parallel$ comes from Equation 5.26:

$$\vec{w} = V^{-1}(\frac{1}{\tilde{e}}) \tag{6.6}$$

So, given estimation times and engine's covariance matrix (which was pre-computed for each training set during cross-validation), $\parallel \tilde{w} \parallel$ was quickly computed and validation set was checked.

Table 6.3 presents the results after cross-validation, following the same match type definitions from last section. As it can be seen, the best engine was selected as the first or second better 72.34% for CEX group and 64.79% for Proven group. In yellow we present the unsatisfactory results, which comprises the selections that do not include the best engine.

This methodology takes also into account the engine's correlation. Therefore, if the two best engines have high correlation, one of them will not be selected. However, if the two best engines are highly correlated, selecting the second best instead of the

Table 6.3. Percentage of time of each kind of match for selection based in execution time and variance.

Kind of match	CEX	Proven
Exact match	18.81%	11.29%
2 matches	12.53%	9.63%
1 match best	41%	43.87%
1 match	24.97%	32.11%
No match	2.69%	3.11%

best should not bring big effect to final result that is to increase the chance to arrive to a proof result. A good hint that this happens is the mean distance between the known best match and the chosen best match for all "1 match" and "No match" matches, for all runs. For "Selection based on engine's time" methodology, this mean distance was 0.73 while for "Selection based on engine's time and correlation", this mean was 0.13, showing a bigger proximity of known best match and chosen best match when correlation is taken into account. If, then, we include the "1 match" values (green line in Table 6.3) to the final percentages, the final results change to 97.31% for CEX group and 96.90% for Proven group.

It is worth noting that since V^{-1} can be precomputed when the training set is built, the whole heuristic performance is very good, taking a very small fraction of the proof setup time, being negligible for practical purposes.

6.5 Summarization

During cross-validation process, linear regression has been computed many times to validate data set. For each time computed, r-squared was summed up and, at the end, the mean was computed. Table 6.4 presents "All data set" r-squared values (already displayed in Table 5.2) and "Mean training set" presents r-squared values obtained after cross-validation.

As it can be seen, even with data partitioning into training set ($\frac{2}{3}$) and validation set ($\frac{1}{3}$), r-squared kept much the same, proving the independence of the data set.

During cross-validation the number of matches for both heuristics ("Selection based in engine's time" and "Selection based in engine's time and correlation") was computed and summarized. Figure 6.2 summarizes results already displayed in Tables 6.2 and 6.3.

It can be seen that the heuristic based just in execution time has the best results for "Exact match" and "2 matches", since this was its objective: order engines by

Table 6.4. Mean r-squared for engine estimators after cross-validation.

	CEX Group		Proven Group	
Engine	**All data set**	**Mean training set**	**All data set**	**Mean training set**
e_1	0.584	0.655	0.516	0.530
e_2	0.573	0.636	0.298	0.310
e_3	0.526	0.594	0.704	0.713
e_4	0.576	0.643	0.366	0.378

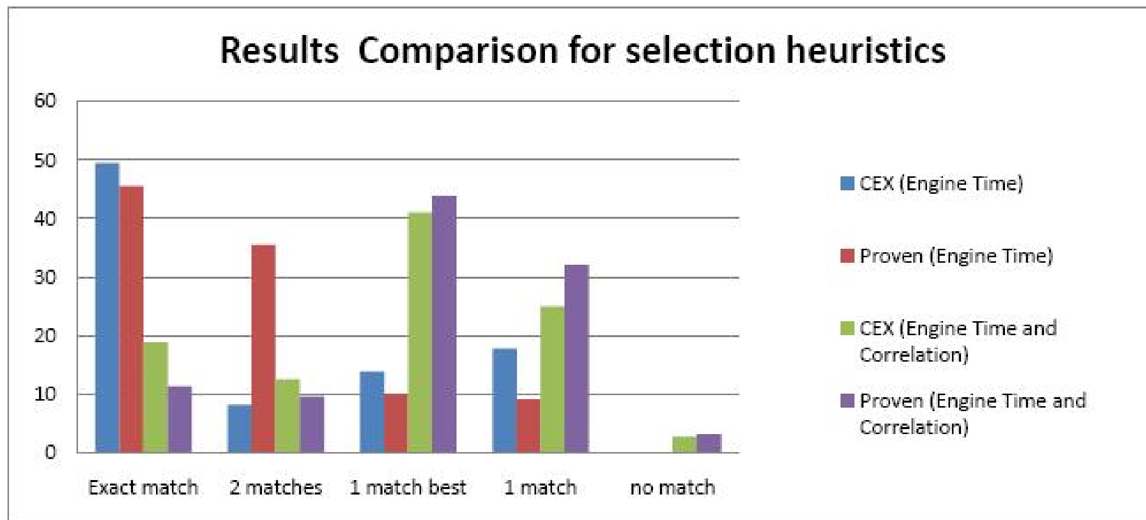

Figure 6.2. Results comparison for selection heuristics.

predicted execution time. In the other hand, maximize chance of success heuristic has the best results for "1 match best" and "1 match", since its objective was to select at least one of the best engines, since if the best two were correlated, they would never be both selected. The significance of the second heuristic would probably be better seen when we have a bigger number of engines.

Chapter 7

Conclusions

Despite all the recent advances in formal verification technology [33], formal verification is an NP-complete problem in the size of the trace description (Binary Decision Diagrams state-explosion problem) and in the verification time (SAT solvers). Therefore, as many other NP-complete problems, practical solutions implement heuristics to try to solve efficiently such problems.

Approaches used today to solve this problem are based mainly on brute force or on verification engineers' feelings. By brute force, a user starts one proof process/thread for each available engine and gets the first result available. However, since engines and associated parameters may be much more numerous than the processing power available, this approach can easily be shown not to scale. On the other hand, verification engineers may have a feeling of the best heuristic to prove a property, but with the increase in design size, trusting in these feelings becomes a big risk.

With the advent of parallel processing (simultaneous use of more than one CPU or processor core to execute a program or multiple computational threads), the possibility of running multiple processes in parallel became more transparent and viable. As a consequence, relying on the use of multiple processes/threads in order to get the best results arose as a way to improve formal verification.

In general, there are a large pool of engines, but only a small number of processing cores. The problem then becomes selecting the best subset of engines that will increase the chance of solving the problem, which in this context means reaching a proof conclusion.

This problem becomes important as there is a large number of heuristics and parameter configurations published over the years to improve the performance of formal verification techniques.

This work presented a methodology to select n engines (heuristics with associated

parameters) out of N ($N >> n$) to run in parallel environments, in order to maximize the chance of reaching a proof conclusion. The methodology consisted of generating performance estimators for engines based in a multivariate statistical model and select the best engines, considering not only the estimated performance of the engines, but also the possibility that some of the engines in a formal verification environment were correlated.

The methodology was based on the computation of good engine's performance estimators. In our case, a significant training set was built, and we collected 15 structural and testability metrics from RTL designs. These metrics ranged from number of flops and gates for structural parameters to SCOAP indexes for the testability parameters.

We initially clustered the training set using k-nearest neighbors in three groups of data (proven valid or True, counterexample, proof timeout). Proof timeout group was ignored since it reflects inconclusive proof. A multivariate linear regression was applied to the other two groups, generating 8 proof time estimators. The estimators relied on polynomial and exponential terms, yielding a good prediction model, based on the Pearson correlation coefficient.

Cross validation is an important mechanism used to validate a prediction model, giving its accuracy. This work repeatedly (1) divided the data set randomly into training data set (66%) and validation data set (33%), (2) generated engine estimators, (3) applied results to validation set and (4) summarized results.

We showed that engine selection could be determined based just on smallest estimation times, but it could also take into account the engine's covariance (by minimizing the engine's correlation), thus maximizing chance of success in selection.

Results confirmed that our methodology was able to provide a very quick selection mechanism for parallelization of engines as, since the inverse of the covariance matrix can be pre-computed when we build the training set, the majority of the time to consider is the metrics data collection for the target to prove, which showed up to be a negligible time when compared to the proof setup time.

Both selection mechanisms showed up as a good heuristic. Selection based on engine's performance showed up very good matching results while "Maximization of the chance of success" selection mechanism showed how significant it can be for environments with a big number of engines, where correlation will help improving the selected group of engines in order to have a diversified group that can range many uncorrelated heuristics and improve the chance of reaching proof conclusion.

Selection based on engine's performance allowed speedups greater than 2 times while "Maximization of the chance of success" selection mechanism allowed speedups greater than 3 times when compared to a not optimal selection.

This work brings a new concept for engine's selection, based on structural and testability metrics, which can be done on-the-fly during proof. Prior works had tried to maximize specific heuristics, for specific designs with specif characteristics. This work can be applied to any design, giving good estimators have been generated.

As future works, we would like to further categorize the proof times based on the number of cycles of the proofs, both for bounded proofs (or timeout) and for proofs where we have a counter-example. In such a scenario, we could create a cluster for bounded / counterexample proofs shorter than 10 cycles, within 10 and 30 cycles, and above 30 cycles, for example. We could also use Bayesian (Conditional Probability) theory, in which, given the user has already spent some effort in trying their proof, whether he/she had better change his/her heuristics or not.

In addition, it is worth investigating new parameters/metrics to generate better engine's time estimators. Increasing training set size is also a way of increasing the quality of estimators.

Bibliography

[1] $http://www.systemverilog.org/$.

[2] Ovl users site. $http://www.eda-stds.org/ovl/$.

[3] e language reference manual. $http://www.ieee1647.org/downloads/prelim_{el}rm.pdf sa = Uei = KsiWUOS0IImg8gTamIHoAgved = 0CBwQFjACusg = AFQjCNH4QhKrpWSmrqi-9DQVvywW5pi6Gw$, 1998-2002.

[4] Ieee 1850 standard for property specification language. $http://www.eda.org/vfv/docs/PSL-v1.1.pdf$, 2004.

[5] Ieee 1666 standard systemc language reference manual. $http://standards.ieee.org/getieee/1666/$, 2011.

[6] Openvera website. $http://www.open-vera.com$, November 2012.

[7] S. B. Akers. Binary decision diagrams. In *IEEE Transactions on Computers*, 1978.

[8] N. Amla, X. Du, A. Kuehlmann, R. P. Kurshan, and K. L. McMillan. An analysis of SAT-based model checking techniques in an industrial environment. In *CHARME*, pages 254–268, 2005.

[9] N. Amla, R. P. Kurshan, K. L. McMillan, and R. Medel. Experimental analysis of different techniques for bounded model checking. In *TACAS*, pages 34–48, 2003.

[10] A. Aziz, S. Tasiran, and R. Brayton. Bdd variable ordering for interacting finite state machines. In *DAC*, 1994.

[11] B. Bailey. A new vision of scalable verification. http://www.eetimes.com/news/design/features/showArticle.jhtml?articleID-=18400907, Mar 2004.

[12] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. In *POPL*, 1981.

[13] R.G. Bennetts. *Design of testable logic circuits.* Addison-Wesley Publishing Company, 1984.

[14] A. Biere. The evolution from limmat to nanosat. Technical report, Dept. of Computer Science, ETH Zurich, 2004.

[15] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS*, 1999.

[16] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. In *IEEE Transactions on Computers*, 1996.

[17] R. S. Boyer and J. S. Moore. *A Computational Logic Handbook.* New York: Academic Press, 1988.

[18] F. Brglez. On testability of combinational networks. In *ISCAS*, pages 221–225, 1984.

[19] R. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, pages C–35–8, 677–691, August 1986.

[20] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with applications to integer multiplication. In *IEEE Transactions on Computers*, 1991.

[21] P. Chauhan, E. M. Clarke, J. Kukula, S. Spra, H. Veith, and D. Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In *FMCAD*, 2002.

[22] G. A. Cherry and S. J. Qin. Multiblock principal component analysis based on a combined index for semiconductor fault detection and diagnosis. In *IEEE Transactions on Semiconductor Manufacturing*, volume 19, pages 159–172, May 2006.

[23] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, 1982.

[24] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, 2000.

[25] J. Cohen, P. Cohen, S. G. West, and L. S. Aiken. *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences.* Lawrence Erlbaum, Associate Publishers, 2003.

[26] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. In *Communications of the ACM*, volume 5, pages 394–397, 1962.

[27] M. Davis and H. Putnam. Computing procedure for quantification theory. In *Journal of ACM*, volume 7, pages 201–214, 1960.

[28] N. Een and N. Sorensson. An extensible SAT-solver. In *Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[29] E. J. Elton and M. J. Gruber. *Modern Portfolio Theory and Investment Analysis*. John Wiley, 1995.

[30] G. Eugene and N. Yakov. Berkmin: A fast and robust SAT-solver. *Discrete Appl. Math.*, 155(12):1549–1561, 2007.

[31] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. In *IEEE Transactions on Computers*, 1990.

[32] M. Ganai and A. Aziz. Improved sat-based bounded reachability analysis. In *VLSI Design*, 2002.

[33] M. Ganai and A. Gupta. *SAT-Based Scalable Formal Verification Solutions*. Springer-Verlag New York Inc, 2007.

[34] M. Ganai and A. Gupta. *SAT-based scalable formal verification solutions*. Springer, 2007.

[35] M. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based umbounded model checking using circuit cofactoring. In *ICCAD*, 2004.

[36] M. Ganai, A. Gupta, and P. Ashar. Lazy constraints and SAT heuristics for proof-based abstractions. In *VLSI Design*, 2005.

[37] M. Ganai, A. Gupta, Z. Yang, and P. Ashar. Efficient distributed SAT and distributed SAT-based bounded model checking. In *CHARME*, 2003.

[38] M. Ganai, L. Zhang, P. Ashar, and A. Gupta. Combining strengths of circuit-based and CNF-based algorithms for a high performance SAT solver. In *DAC*, 2002.

[39] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli. DPLL(T): Fast decision procedures. In *CAV*, 2004.

[40] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[41] E. Goldberg and Y. Novikov. Berkmin: A fast and robust SAT-solver. In *DATE*, 2002.

[42] L. H. Goldstein and E. L. Thigpen. SCOAP: Sandia controllability/observability analysis program. In *25 years of DAC: Papers on Twenty-five years of electronic design automation*, pages 397–403, New York, NY, USA, 1988. ACM.

[43] J. Grason. TMEAS, a testability measurement program. In *16th Conference on Design automation*, pages 156–161, San Diego, CA, United States, June 1979.

[44] A. Gupta, M. Ganai, P. Ashar, and Z. Yang. Iterative abstraction using SAT-based BMC with proof analysis. In *ICCAD*, 2003.

[45] A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar. Abstraction and BDDs complement in SAT-based BMC in DiVer. In *CAV*, 2003.

[46] A. Gupta, M. Ganai, C. Wang, Z. Yang, and P. Ashar. Learning BDDs in SAT-based bounded model checking. In *DAC*, 2003.

[47] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[48] J. Hergov and C. Meinel. Efficient boolean manipulation with OBDDs can be extended to FBDDs. In *IEEE Transactions on Computers*, 1994.

[49] G. J. Holzmann. The model checker SPIN. In *IEEE Transactions on Software Engineering*, 1997.

[50] R. Hum. Static verification needs a parallel approach. http://www.eetimes.com/news/design/columns/eda/showArticle.jhtml?articleID=21800552, Jun 2004.

[51] F. Hutter, D. Babic, H. H. Hoos, and A. J. Hu. Boosting verification by automatic tuning of decision procedures. In *FMCAD '07: Proceedings of the Formal Methods in Computer Aided Design*, pages 27–34, Washington, DC, USA, 2007. IEEE Computer Society.

[52] S. K. Jain and V. D. Agrawal. STAFAN: An alternative to fault simulation. In *21st Des. Autom. Conf.*, pages 18–23, 1984.

[53] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[54] R. P. Kurshan. *Computer-Aided Verification of Co-ordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.

[55] J. P. Marques-Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. In *IEEE Transactions on Computers*, volume 48, pages 506–521, 1999.

[56] J. P. Marques-Silva and K. A. Sakallal. GRASP: A search algorithm for propositional satisfiability. In *IEEE Transactions on Computers*, volume 48, pages 506–521, 1999.

[57] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic Publishers, 1993.

[58] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *CAV*, pages 250–264, 2002.

[59] K. L. McMillan. Interpolation and sat-based model checking. In *CAV*, 2003.

[60] K. L. Mcmillan and N. Amla. Automatic abstraction without counterexamples. In *TACAS*, pages 2–17. Springer, 2003.

[61] S. Minato. Zero-supressed BDDs for set manipulation in combinational problems. In *DAC*, 1993.

[62] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *DAC*, 2001.

[63] A. Nadel. *Backtrack Search Algorithms for Propositional Logic Satisfiability: Review and Innovations.* PhD thesis, Hebrew University of Jerusalem, 2002.

[64] A. Narayan, A. Jain, M. Fujita, and A. L. Sangiovanni-Vincentelli. Partitioned-ROBDDs â a compact, canonical and efficiently manipulable representation for boolean functions. In *ICCAD*, 1996.

[65] J. Pilarski and G. Hu. SAT with partial clauses and back-leaps. In *DAC*, 2002.

[66] A. Pnueli. The temporal logic of programs. In *IEEE Symposium on Foundation of Computer Science*, 1977.

[67] P. H. Qinghua. Fault detection using principal component based k-nearest-neighbor rule. In *AIChe 2007 Annual Meeting*, Salt Lake City, November 2007.

[68] R. Ranjan, J. Sanghavi, R. Brayton, and A. L. Sangiovanni-Vincentelli. High performance BDD package based on exploiting memory hierarchy. In *DAC*, 1996.

[69] I. Ratiu, A. Sangiovanni-Vincentelli, and D. Pederson. VICTOR: A fast vlsi testability analysis program. In *International Test Conference*, Philadelphia, USA, November 1982.

[70] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and decomposition of decision diagrams. In *DAC*, 1998.

[71] R. Rudell. Variable ordering for binary decision diagrams. In *ICCAD*, 1993.

[72] S. C. Seth, L. Pan, and V. D. Agrawal. PREDICT–probabilistic estimation of digital circuit testability. In *Fault-Tolerant Comput. Symp (FTCS-15)*, pages 220–225, 1985.

[73] M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT solver. In *FMCAD*, 2000.

[74] J. P. M. Silva and K. A. Sakallah. GRASP: a new search algorithm for satisfiability. In *International Conference on Computer-Aided Design*, 1996.

[75] L. I. Smith. A tutorial on principal component analysis. $http : //csnet.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf$, Feb 2002.

[76] F. Somenzi. CUDD: CU decision diagram package release 2.2.0, 1998.

[77] H. Touati, H. Savoj, B. Lin, R. Brayton, and A. L. Sangiovanni-Vincentelli. Implicit state ennumeration of finite state machines using BDDs. In *ICCAD*, 1990.

[78] J. Yuan, C. Pixley, and A. Aziz. *Constrained-Based Verification.* Springer, 2006.

[79] H. Zhang. SATO: An efficient propositional prover. In *International Conference on Automated Deduction*, pages 272–275, July 1997.

[80] H. Zhang. SATO: An efficient propositional prover. In *14th International Conference on Automated deduction*, volume 1249, 1997.

[81] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *IEEE/ACM international conference on Computer-aided design*, pages 279–285. IEEE Press, 2001.

[82] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *CAV*, 2002.

A

Testability measures

The continuing development of the manufacturing technology has lead to an increase in the density of gates possible on integrated circuits, with the result that access to their internal nodes is mode difficult. Therefore, there is a growing concern about the ease of testability of these integrated circuits.

Testability analysis evaluates the relative degree of difficulty of testing circuit nodes for line stuck-at faults. Testability analysis depends on two properties: controllability and observability.

Controllability is a measure of the ease or difficulty of setting a node to a desired logic value, while observability is a measure of the ease or difficulty of propagating a node‘s value to a primary output. A node is said to be easily testable if it is highly controllable and observable.

A fault is testable if there exists a well-specified procedure to expose it, which is implementable with a reasonable cost using current technologies. A circuit is testable with respect to a fault set when each and every fault in this set is testable.

The main goals of testability are:

- Maximize fault coverage: [(no. of faults detected)/(no. of possible faults)] x 100
- Minimize test application time
- Minimize test data volume
- Minimize test generation (ATPG) effort
- Maximize fault resolution (isolating fault to smallest replaceable component)
- Minimize hardware/software overhead for testing

Several testability measures have been suggested in the literature to assist in the process of testing circuits (SCOAP [42], COP [18], PREDICT [72], VICTOR [69], STAFAN [52], TMEAS [43], CAMELOT [13]).

In the scope of this work, testability measures can give information about a design or property that may be useful to decide upon an engine.

A.1 SCOAP (Sandia Controllability Observability Analysis) [42]

A testability measure such as SCOAP gives integral numerical estimates of the controllability and observability of signal lines in a given circuit.

SCOAP characterizes circuit nodes as sequential or combinational. A combination node is defined to be a primary input or a combinational standard cell output, while a sequential node is an output node of a sequential standard cell. SCOAP represents the testability of each node N by a vector having six elements:

1. CC0(N) combinational 0-controllability,
2. CC1(N) combinational 1-controllability,
3. SC0(N) sequential 0-controllability,
4. SC1(N) sequential 1-controllability,
5. CO(N) combinational observability,
6. SO(N) sequential observability.

CC0(N) and CC1(N) are the minimum number of combinational node assignments required to set node N to "0" or "1", respectively. Analogously, SC0(N) and SC1(N) are the minimum number of sequential node assignment required to set node N to "0" or "1", respectively. CO(N) is the number of combinational standard cells between node N and primary output plus the minimum number of combinational node assignments required to propagate the logical value on node N to a primary output of the circuit. Similarly, SO(N) is the number of sequential standard cells that must be controlled to propagate the logical value on node N to a primary circuit output. Note that these measures are exact integers that depend on circuit topology.

Some examples of controllability equations are presented in Figure A.1.

Some examples of observability equations are presented in Figure A.2.

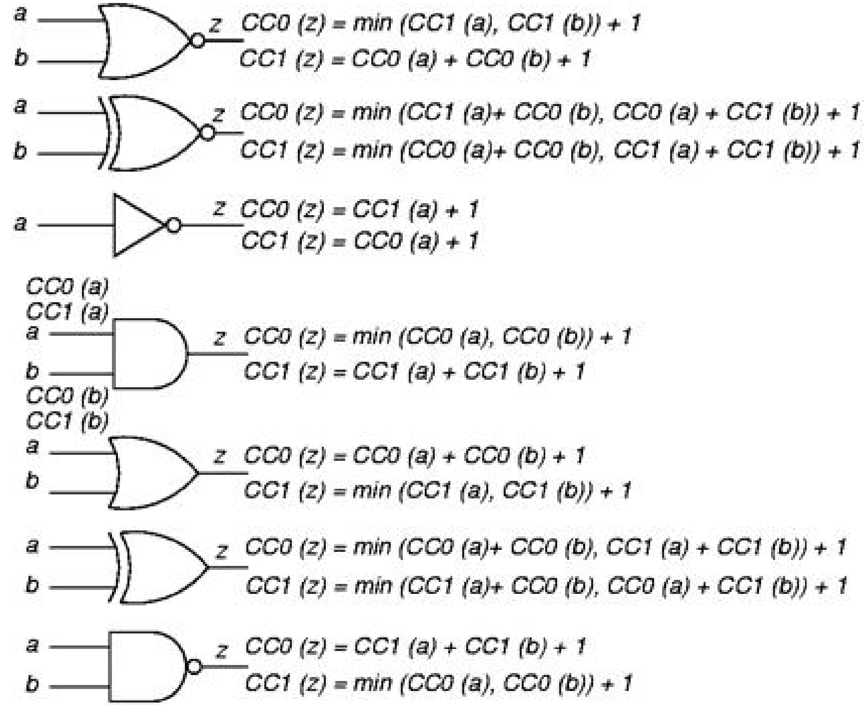

Figure A.1. SCOAP controllability equations.

The main weakness of SCOAP is the assumption that controlling/observing a line is independent event, what is not true in reconvergent paths like below. This limits the accuracy of SCOAP.

Flip-Flop equations based in Figure A.4 are presented below:

$$CC1(Q) = CC1(D) + CC1(C) + CC0(C) + CC0(RESET) \tag{A.1}$$

$$SC1(Q) = SC1(D) + SC1(C) + SC0(C) + SC0(RESET) + 1 \tag{A.2}$$

$$CC0(Q) = \min CC1(RESET) + CC1(C) + CC0(C), CC0(D) + CC1(C) + CC0(C) \tag{A.3}$$

$$CO(D) = CO(Q) + CC1(C) + CC0(C) + CC0(RESET) \tag{A.4}$$

SC0(Q) is analogous to SC1 and SO(D) is analogous to CO(D).

The sequential controllability gives a rough measure of the number of times various flip-flops must be clocked to control a signal and the sequential observability

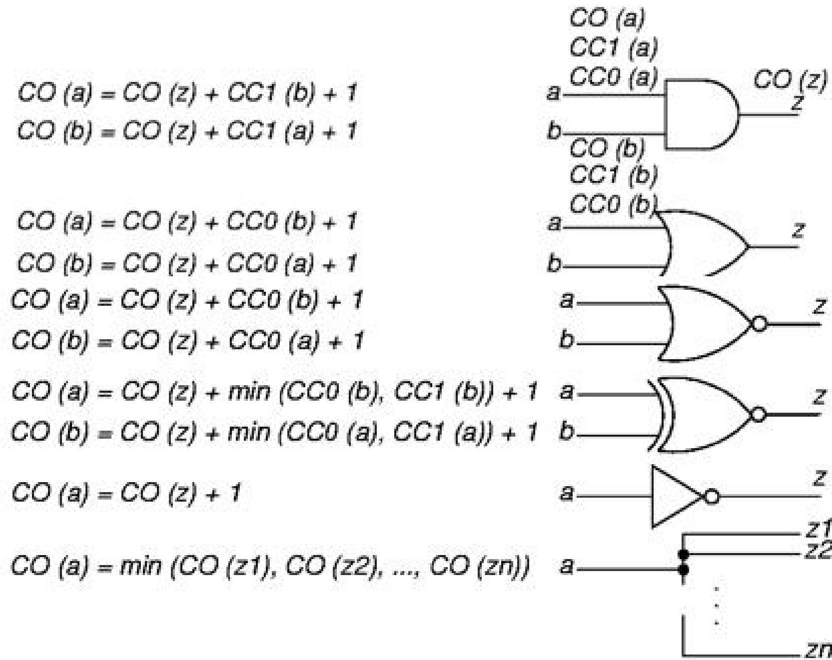

Figure A.2. SCOAP observability equations.

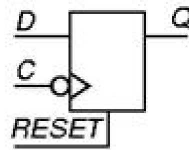

Figure A.3. Flip-flop.

measures the number of times various flip-flops must be clocked to observe a signal. Generally, these sequential measures characterize the test length.

SCOAP may also be used to predict the length of the test vector set for a circuit. The testability of the stuck-at faults at node x are defined as:

$$T(x, stuck-at-0) = CC1(x) + CO(x) \tag{A.5}$$

$$T(x, stuck-at-1) = CC0(x) + CO(x) \tag{A.6}$$

$$TestabilityIndex = log \sum_{all f_j} T(f_j) \tag{A.7}$$

In order to detect a fault at x, one must set x to the opposite value from the fault and observe x at a PO. Figure A.4 shows a linear relationship between the number of vectors needed for 90% fault coverage and the Testability index for a number of

circuits.

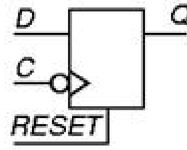

Figure A.4. SCOAP observability equations.

The computation of SCOAP testability measures is improved by a new testability analysis tool named "HISCOAP". It is a hierarchical testability analysis tool which makes use of the hierarchy of the circuit. The tool significantly reduces the memory and computational resources while computing testability measures.

A.2 COP (Controllability/Observability Program) [18]

COP uses probability theory to calculate testability measures. It assigns three probability values to each line:

1. C1(l): the 1-controllability of a line l is defined as the probability that line l takes value "1".

2. C0(l): the 0-controllability of a line l is defined as the probability that line l takes value "0".

3. O(l): the observability of line l is the probability of observing the value of line l at a primary output.

COP is a simple direct application of probability theory for the calculation of testability.

A.3 PREDICT [72]

PREDICT breaks the circuit into sub-circuits called supergates. The supergates completely include reconvergent fanouts, what enhances measure quality. The worst case would be the entire circuit as a supergate.

PREDICT computes exact probabilities. Computational complexity is exponential with the circuit size. Several heuristics were developed to reduce calculations time.

A.4 VICTOR [69]

VICTOR is different from the other testability programs in its emphasis on detecting redundant stuck-at faults. VICTOR is restricted to combinational circuits.

It defines three controllability and three observability parameters for every node in the circuit: label, weight and size. A label is a sufficient test for controlling or observing a node. The weight is the weighted sum of the constrained primary inputs and fan-out branches associated with a certain label.

In addition to calculating testability measures for every node and identifying redundant faults, VICTOR generates test vectors for some of the faults in the circuit. Unfortunately, it is typical in VICTOR to identify many nodes as being potentially redundant even if no redundant faults are present.

A.5 STAFAN (Statistical Fault Analysis) [52]

STAFAN also makes use of concepts of controllability and observability. These quantities are redefined as probabilities of controlling and observing the lines. Controllability of a line is estimated by collecting the statistics of activity on that line. Observability is then computed from the estimated controllability. The product of the appropriate controllability and observability gives the detection probability of a fault.

STAFAN uses probability theory in a way similar to COP. STAFAN, however, assigns two observability values to each node: 1-observability and 0-observability. These observability values are defined similarly to COP's observability, with the added initial condition that line l is set to “1” or “0”, respectively. STAFAN calculates the controllability values based on an experiment that uses random vectors, while COP calculates them based on the Boolean equations of each node. This makes the testability measures calculated by COP dependent only on the circuit, while STAFAN produces testability measures that depend on both the circuit and the test vectors used in the experiment. Hence, STAFAN produces different values for testability measures based on the test vectors used in the experiment, while COP values are always consistent.

A.6 TMEAS (Testability Measure Program) [43]

This approach was developed for register-transfer-level (RTL) circuits, but can also be applied at the gate level. The measures are normalized between “0” and “1” to reflect the ease of controlling and observing the internal nodes. The approach is summarized as follows:

1. For each signal line s, we denote the controllability of s as CY(s) and the observability of s as OY(s).

2. The values for the CYs and the OYs of all the signal lines are derived by solving a system of simultaneous equations with the CYs and the OYs as unknowns.

   a) Let the input variables of a component be $x_1, x_2, \ldots, x_n$, and the output variables be $z_1, z_2, \ldots, z_m$. The expression used to calculate CY for each output z_j is:

   $$CY(z_j) = CTF \times \frac{1}{n}\sum_{i=1}^{n} CY(x_i) \tag{A.8}$$

   where CTF is the controllability transfer factor of the component.

   b) Let $N_j(0)$ and $N_j(1)$ be the numbers of input combinations for which z_j has value 0 and 1, respectively. Then

   $$CTF = \frac{1}{m}\sum_{i=1}^{m}(1 - \frac{|N_i(0) - N_i(1)|}{2^n}), 0 \leq CTF \leq 1 \tag{A.9}$$

   Note that $N_j(0) + N_j(1) = 2$n and, for a given primitive, each output controllability is assigned the same value.

   c) The expression used to calculate OY for each input x_i is

   $$OY(x_i) = OTF_x \frac{1}{m}\sum_{j=1}^{m} OY(Z_j) \tag{A.10}$$

   Where OTF is the observability transfer factor of the component.

   d) Let NS_i be the numbers of input combinations for which the change of x_i results in a change of output. Then

   $$OTF \equiv \frac{1}{n}\sum_{i=1}^{n}\frac{NS_i}{2^n} \tag{A.11}$$

   NS_i also means the number of input combinations that can sensitize a path from x_i to the output. OTF measures the probability that a faulty value at any input will propagate to the outputs. Note that $0 \leq OTF \leq 1$. For a given primitive, each input observability is assigned the same value.

3. Fanouts: Let s be a fanout stem and k be the number of its branches. Then the CYs of each fanout branch is

$$CY = \frac{CY(s)}{(1 + logk)} \tag{A.12}$$

The observability of the fanout stem s is

$$OY(s) = 1 - \prod_{i=1}^{k}[1 - OY(b_i)] \tag{A.13}$$

Where b_i are fanout branches of s.

4. Sequential components: Sequential components are modeled by adding back links around the components that represent internal states.

A.7 CAMELOT (Computer-Aided Measure for Logic Testability) [13]

The Computer-Aided MEasure for LOgic Testability (CAMELOT) was intended to be an improvement on TMEAS discussed above.

CAMELOT assigns a single controllability value, $CY \in [0, 1]$, to every gate-level line in the circuit. The maximum value 1 indicates a node, such as a primary input, where forcing a "1" is as easy as forcing a "0". At the other extreme, a CY of 0 indicates a node that cannot be set to "1" or "0". In practice, the majority of the nodes in a circuit have CY node values between these two limits.

Similarly, CAMELOT assigns a value OY to each node indicating the observability of that node. OY is defined to be a measure of the ease of observing a state of a node at the circuit's primary outputs.

The controllability and observability values calculated by CAMELOT are heuristic figures of merit having no mathematical foundation. Clearly, with just one controllability value and one observability value for each node, CAMELOT requires less computation than SCOAP. CAMELOT provides a testability measure for each node:

$$TY_{node} = CY_{node} * OY_{node} \tag{A.14}$$

And a testability measure for the whole circuit:

$$TY_{circuit} = \frac{\sum TY_{nodes}}{Numberofnodes} \tag{A.15}$$