# ANYTIME ALGORITHMS FOR ROBDD SYMMETRY DETECTION AND APPROXIMATION

a thesis submitted to
The University of Kent at Canterbury
in the subject of computer science
for the degree
of doctor of philosophy.

By
Neil J. Kettle

# Contents

# List of Tables

# List of Figures

# Abstract

Reduced Ordered Binary Decision Diagrams (ROBDDs) provide a dense and memory efficient representation of Boolean functions. When ROBDDs are applied in logic synthesis, the problem arises of detecting both classical and generalised symmetries. State-of-the-art in symmetry detection is represented by Mishchenko's algorithm. Mishchenko showed how to detect symmetries in ROBDDs without the need for checking equivalence of all co-factor pairs. This work resulted in a practical algorithm for detecting all classical symmetries in an ROBDD in $O(|G|^3)$ set operations where $|G|$ is the number of nodes in the ROBDD. Mishchenko and his colleagues subsequently extended the algorithm to find generalised symmetries. The extended algorithm retains the same asymptotic complexity for each type of generalised symmetry. Both the classical and generalised symmetry detection algorithms are monolithic in the sense that they only return a meaningful answer when they are left to run to completion. In this thesis we present efficient anytime algorithms for detecting both classical and generalised symmetries, that output pairs of symmetric variables until a prescribed time bound is exceeded. These anytime algorithms are complete in that given sufficient time they are guaranteed to find all symmetric pairs. Theoretically these algorithms reside in $O(n^3 + n|G| + |G|^3)$ and $O(n^3 + n^2|G| + |G|^3)$ respectively, where $n$ is the number of variables, so that in practice the advantage of anytime generality is not gained at the expense of efficiency. In fact, the anytime approach requires only very modest data structure support and offers unique opportunities for optimisation so the resulting algorithms are very efficient. The thesis continues by considering another class of anytime algorithms for ROBDDs that is motivated by the dearth of work on approximating ROBDDs. The need for approximation arises because many ROBDD operations result in an ROBDD whose size is quadratic in the size of the inputs. Furthermore, if ROBDDs are used in abstract interpretation, the running time of the analysis is related not only to the complexity of the individual ROBDD operations but also the number of operations applied. The number of operations is, in turn, constrained by the number of times a Boolean function can be weakened before stability is achieved. This thesis proposes a widening that can be used to both constrain the size of an ROBDD and also ensure that the number of times that it is weakened is bounded by some given constant. The widening can be used to either systematically approximate an ROBDD from above (i.e. derive a weaker function) or below (i.e. infer a stronger function). The thesis also considers how randomised techniques may be deployed to improve the speed of computing an approximation by avoiding potentially expensive ROBDD manipulation.

# Acknowledgements

I would like to thank Andy King, a very understanding, and not to mention entertaining supervisor. I will miss our interesting and diverse conversation. Secondly, I would like to thank my supervisory panel members David Shrimpton and Keith Hanna.

Further, I thank Tadeusz Strzemecki, Jacob Howe, Laurent Mauborgne, Axel Simon, Peter Schachte, Harald Søndergaard and Jin Zhang for useful discussions throughout my time working on this thesis.

The work presented herein was funded by EPSRC Grant EP/C015517, British Council Grant PN 05.021 and the Royal Academy of Engineering Grant IJB/PS/ITG 05–1046.

# Chapter 1

# Introduction

> "A computer lets you make more mistakes faster
> than any invention in human history - with the
> possible exceptions of handguns and tequila."
> - M. Ratcliffe

The notion of an *anytime* algorithm is one borne of the acceptance of the intractability of algorithms whose runtime is bounded even polynomially. In complexity theoretic terms, an algorithm is deemed efficient (or tractable) if the time required to solve every instance of the underlying problem is bounded polynomially in the size of the input. This is a valid assumption from a complexity theoretic standpoint in the presence of the Church-Turing thesis but has little practical importance when faced with a polynomial time algorithm requiring $O(n^{100})$ time! The problem is further compounded if the only known algorithms are *monolithic*. For such an algorithm one has to assume that inputs exist such that the worst-case behaviour of the algorithm will be exhibited (either in terms of space or time). In such a scenario, we have no choice but to allow the algorithm to run to completion before any meaningful, or correct results can be obtained. Consequently, if the problem we wish to solve is computationally hard, we may be forced to wait an unacceptable amount of time for an answer to appear.

The purpose of an anytime algorithm is to allow the user, or the algorithm itself, to terminate early should the computation exhaust some predefined resource bound. The seminal paper on the anytime approach defined an algorithm as being anytime if:

> "it can be terminated at any time ... and the answers returned improve in some well-behaved manner as a function of time." [DB88]

The key requirement of an anytime algorithm is that early termination should not compromise the correctness of the computed result. In this respect, an anytime algorithm can be thought of as an approximation algorithm in that the result is guaranteed to be "correct" (with absolute *certainty*) irrespective of the computational resources available.

However, the result may not be "complete" in the sense that it is the most accuracte result obtainable.

Although anytime algorithms have not received much recognition in the academic literature outside of the engineering community [FL97, RB04, KFM06], the anytime concept occurs naturally in several common types of algorithm, most notably *iterative approximation* algorithms [TW69], *one-sided error randomised* algorithms [GS05] and *approximation algorithms* for some NP-hard optimisation problems [GCA95].

Zilberstein [Zil96] reviews various anytime algorithms and suggests that ideally an anytime algorithm should possess the following desirable properties.

- the quality of an intermediary approximate answer can be determined whilst the algorithm continues to execute – *recognisability*.
- the quality of the results increases monotonically with time – *monotonicity*.
- the improvement in intermediary result quality is larger in early stages of computation and diminishes over time – *diminishability*.
- the algorithm may be stopped at any time without compromising the correctness of the answer – *interruptibility*.

This thesis reports novel anytime algorithms for *Reduced Ordered Binary Decision Diagram* (ROBDD) manipulation that satisfy the above properties.

## 1.1 Research Statement and Motivation

One may think that the development of anytime algorithms for ROBDD manipulation is merely an exercise in algorithmic aesthetics, however, in practise the advantages of the anytime approach are often mandated by problems in engineering. This study was no exception, indeed, it was motivated by the desire to develop program analysis techniques for the automated detection of security vulnerabilities. Therefore in order to motivate the body of the thesis and to put the work into the context, this section outlines the construction, and subsequent implementation, of a program analysis technique to discover integer overflow vulnerabilities. The anytime algorithms were motivated by problems encountered during the development of the analysis. To summarise, this section details the motivating factors that inspired the research presented herein.

### 1.1.1 Motivating Example

An integer is a variable capable of representing any number with no fractional part. For efficiency, integers are normally the same size as the word-width of the underlying system (i.e. 32 bits on a 32 bit Intel architecture, and 64 bits on an ALPHA/SPARC microprocessor). Since any integer is of a fixed size (assume 32 bits), there is a fixed

maximum value it can store ($2^{32} - 1$ for an unsigned integer of 32 bits in size). Any computation whose result is larger than this maximum value is said to cause an integer overflow. The effect integer overflows have is somewhat mitigated in languages providing automated array bounds checking (Java and Pascal). However in languages such as C, that do not provide any form of bounds checking, an integer overflow results in "undefined behaviour" (ISO C99 standard [ISO99]). Hence, compilers conforming to the [ISO C99] standard may do anything from ignoring the overflow to aborting the program. However, most compilers ignore the overflow, resulting in an unexpected value being stored. This is often due to the complexities involved in detecting integer overflows in hardware, not to mention the efficiency implications of verifying the correctness of the results of all computations.

There are several security implications associated with integer overflows. Primarily, the problem is that integer overflows cannot be detected after they have occurred, so there is no obvious way for an application to tell if the result of an earlier calculation has indeed caused an overflow. Moreover, ISO C99 states "A computation involving unsigned operands can never overflow, because a result that cannot be represented by the resulting unsigned integer type is reduced modulo the number that is one greater than the largest value that can be represented by the resulting type." [ISO99]. Although this interpretation of unsigned arithmetic is self-consistent, it is not necessarily consistent with the model that a programmer has for arithmetic. This is because integer operations are computed modulo $2^n$, hence there exists a possibility for a mismatch between the value computed and what the programmer originally intended.

An integer overflow on its own is not a security vulnerability, indeed, it is perfectly legitimate for any algorithm to utilise modular reduction. For example, many modern symmetric block cryptographic algorithms utilise integer arithmetic and modular reduction to implement non-linear transforms [LM91]. However, an integer overflow can be hazardous if the calculation involved is in some way associated with the size of a buffer or an index for an array access as an overflow here may not be intended.

An example of an integer overflow can be found in Figure 1, which illustrates how integer overflows occur in programs written in the C programming language [Meh02]. As we may observe, should the value of `*sizep` and `elsize` be such that the calculation `c` $*$ `elsize` results in `nodesize` taking a value such that `nodesize` $\geq 2^{32}$ at line #21, then an integer overflow will occur resulting in an incorrect number of bytes being allocated at line #28 since the program assumes that at least `c` $*$ `elsize` bytes have been allocated. Furthermore, this condition will result in a buffer overflow vulnerability when an attempt is made to write `c` $*$ `elsize` bytes to the newly allocated buffer.

In addition to the disclosed vulnerabilities that have been previously reported, the ubiquity of unchecked arithmetic suggests that many vulnerabilities still reside in code that has been subject to a code audit. One such denial of service (DoS) vulnerability that

```
[1]  bool_t
[2]  xdr_array (xdrs, addrp, sizep, maxsize, elsize, elproc)
[3]       XDR *xdrs;
[4]       caddr_t *addrp;        /* array pointer                    */
[5]       u_int *sizep;          /* number of elements               */
[6]       u_int maxsize;         /* max numberof elements            */
[7]       u_int elsize;          /* size in bytes of each element    */
          ...
[8]  {
[9]      caddr_t target = *addrp;
[10]     u_int nodesize, c;
         ...
[11]     /* like strings, arrays are really counted arrays */
[12]     if (!xdr_u_int (xdrs, sizep))
[13]     {
[14]         return FALSE;
[15]     }
[16]     c = *sizep;
[17]     if ((c > maxsize) && (xdrs->x_op != XDR_FREE))
[18]     {
[19]         return FALSE;
[20]     }
[21]     nodesize = c * elsize;
         ...
[22]     if (target == NULL)
[23]         switch (xdrs->x_op)
[24]         {
[25]             case XDR_DECODE:
[26]                 if (c == 0)
[27]                     return TRUE;
[28]                 *addrp = target = mem_alloc (nodesize);
[29]                 if (target == NULL)
[30]                 {
[31]                     (void) fprintf (stderr,
[32]                                     "xdr_array: out of memory\n");
[33]                     return FALSE;
[34]                 }
[35]                 __bzero (target, nodesize);
         ...
[36] }
```

Figure 1: A fragment of xdr_array.c taken from glibc version 2.1.2

stems from an integer overflow is outlined in Figure 2. The figure details a vulnerability found in the MailEnable IMAPv4 service [MEL06] (MEIMAPS.exe). Since MailEnable is a closed-source software product, only assembler output is given. The vulnerability is located in the call to `_memcpy` at line #`10007FFD` and is caused by an integer overflow that occurs whilst calculating the starting point from which memory will be copied. The integer overflow itself is located at line #`10007FD2`, the integer calculated in the register `ecx` is assumed to be a signed integer and therefore has a maximum value of $2^{31} - 1$. Hence, if the result in the register `ecx` is greater than `7fffffffh` $= 2^{31} - 1$ the result will be negative, thus the comparison at line #`10007FD4` will fail since it is possible that `ecx` is negative and thus smaller than `4000h` = 16384. The resultant call to `_memcpy` at line #`10007FFD` will thereby attempt to access non-existent memory and thus cause the application to crash with an 'Access Violation' as shown in Figure 3.

Program analysis seems attractive for discovering these vulnerabilities because it offers a way of systematically covering all paths and behaviours in a program. There are two distinct ways we can do this, the first being to automatically prove the non-existence of an overflow [CCF+05] and hence a modular reduction vulnerability.

The other approach is to prove the existence of *at least* one vulnerability by generating an input such that the program exhibits anomalous behaviour, that is generate a counter example to the correctness of the program. This problem is simpler because we do not have to prove correctness over all flows. Therein lies the power of the method: the generation of an input that causes anomalous behaviour is extremely useful as it provides a simple method to practically demonstrate the vulnerability to a third party.

We envisage that such an analysis for C programs be based upon several phases. The first converts the program into a simply typed language permitting only single assignments to be made to each variable along a control flow path. Further, the value of an integer variable present in a program can be described by a collection of propositional variables, one for each bit of the underlying representation. The semantics of a statement can be modelled with propositional constraints, for instance a statement such as `x := y + z` over 32-bit integers can be described as a system of constraints that relate the values of each of the bits in `x` (after the assignment) to each of the input bits of `y` and `z`. The semantics of sequences of statements can be computed by combining systems of propositional constraints. It is likewise possible to derive a propositional formula in terms of `y` and `z` thats expresses whether the value of `x` has become tainted, that is, compromised by the presence of an overflow. Then the problem of locating modular reduction vulnerabilities reduces to the problem of finding assignments to the propositional variables such that the variable that feeds into a memory allocation operation is tainted.

A natural way of representing a set of propositional constraints is as an ROBDD, however, this leads to a series of computational problems that motivated the work

```
.text:10007F78 public NTLM_UnPack_Type1
.text:10007F78
.text:10007F78 arg_0= dword ptr  8
.text:10007F78 arg_4= dword ptr  0Ch

.text:10007F78  push    ebp
.text:10007F79  mov     ebp, esp
.text:10007F7B  push    20h
.text:10007F7D  mov     eax, [ebp+arg_0]
.text:10007F80  push    eax
.text:10007F81  mov     ecx, [ebp+arg_4]
.text:10007F84  add     ecx, 2000h
.text:10007F8A  push    ecx
.text:10007F8B  call    _memcpy
.text:10007F90  add     esp, 0Ch
.text:10007F93  mov     edx, [ebp+arg_4]
.text:10007F96  movsx   eax, word ptr [edx+2010h] ; 0x00000001
.text:10007F9D  test    eax, eax
.text:10007F9F  jle     short loc_10008005        ; branch not taken
.text:10007FA1  mov     ecx, [ebp+arg_4]
.text:10007FA4  movsx   edx, word ptr [ecx+2010h] ; 0x00000001
.text:10007FAB  cmp     edx, 400h
.text:10007FB1  jge     short loc_10008005        ; branch not taken
.text:10007FB3  mov     eax, [ebp+arg_4]
.text:10007FB6  cmp     dword ptr [eax+2014h], 0  ; 0x7fffffff
.text:10007FBD  jle     short loc_10008005        ; branch not taken
.text:10007FBF  mov     ecx, [ebp+arg_4]
.text:10007FC2  movsx   edx, word ptr [ecx+2010h] ; 0x00000001
.text:10007FC9  mov     eax, [ebp+arg_4]
.text:10007FCC  mov     ecx, [eax+2014h]          ; 0x7fffffff
.text:10007FD2  add     ecx, edx                  ; integer overflow,
                                                  ;   0x00000001+0x7fffffff
                                                  ;   = 0x80000000 = -2147483648
.text:10007FD4  cmp     ecx, 4000h
.text:10007FDA  jge     short loc_10008005        ; branch not taken
                                                  ;   0x80000000 = -2147483648
.text:10007FDC  mov     edx, [ebp+arg_4]
.text:10007FDF  movsx   eax, word ptr [edx+2010h]
.text:10007FE6  push    eax                       ; argument 3: size
.text:10007FE7  mov     ecx, [ebp+arg_4]
.text:10007FEA  mov     edx, [ebp+arg_0]
.text:10007FED  add     edx, [ecx+2014h]          ; argument 2:
                                                  ;   source (invalid)
.text:10007FF3  push    edx
.text:10007FF4  mov     eax, [ebp+arg_4]
.text:10007FF7  add     eax, 1400h
.text:10007FFC  push    eax                       ; argument 1:
                                                  ;   destination
.text:10007FFD  call    _memcpy
```

Figure 2: A fragment of MENTLM.dll from MailEnable Professional v2.33

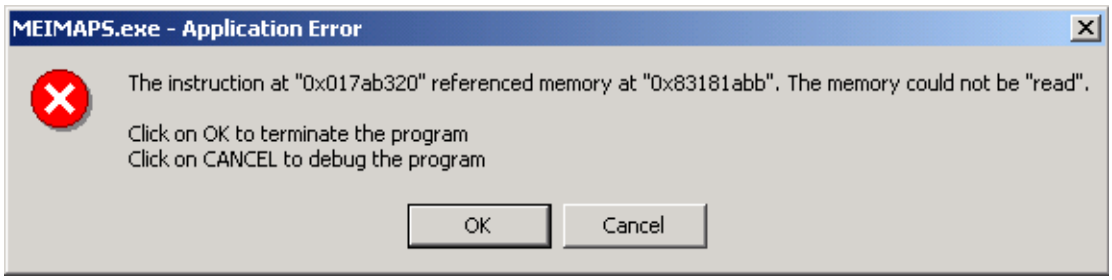


Figure 3: Access Violation in MENTLM.dll from MailEnable Professional v2.33

presented in this thesis.

### 1.1.2 Computational Problems with ROBDDs

The size of an ROBDD critically depends on a number of factors, most notably: the number of propositional variables over which the function is defined; the particulars of the Boolean function that is represented; and the variable ordering used to construct the ROBDD. The problems of ROBDD explosion are exacerbated by the requirement to model a single integer variable by as many as 64 propositional variables. Computing an optimal variable ordering, that is one that minimises the size of an ROBDD, can be expensive [Weg00], but it has long been known that symmetry information can be used to improve variable ordering [MMW93]. This motivated the application of symmetry detection methods, however, the state-of-the-art method [Mis03] was found to be prohibitively expensive on large ROBDDs [KK06]. This suggested the design of an anytime symmetry detection algorithm that may be applied to large ROBDDs to finesse the possibility of intractability.

The optimal variable ordering for many circuits is an interleaved variable ordering in which the propositional variables that share the same offset within a word are grouped contiguously [Bry92]. However, there are some circuits for which the interleaved variable ordering is sub-optimal. Sub-optimality occurs, for instance, when the interleaved variable ordering is used to model the equivalent statements `x := y` $\ll$ `1`, `x := y + y`, and `x := 2*y`. Although an optimal ordering can be found for these statements, the resulting ROBDD cannot be efficiently combined with that of another statement that requires the standard interleaved variable ordering. Bryant *et al.* [BCM$^+$92] discussed this phenomenon, concluding that a *Decision Diagram* will quickly become unmanageable if it needs to model shift-like behaviour. Unfortunately, shifts routinely occur in C programs for efficiency purposes which suggests that ROBDD approximation is vital when modelling programs at the bit level. Unfortunately, state-of-the-art approximation techniques were found to be wanting, in that they are hard to control and frequently returned draconian (grossly), inaccurate approximations [KKS06]. This motivated the

construction of an anytime approximation scheme whose accuracy can be throttled.

### 1.1.3 Further Motivation and Background

The new anytime algorithms presented in this thesis have application outside the arena of integer overflow vulnerability detection. In particular, these algorithms find application in abstract interpretation and logic synthesis. An overview of these applications is given below.

#### Abstract Interpretation

Abstract interpretation is a theory of abstract semantics and approximation which is used for the construction of semantics-based program analysis techniques [CC77,CC92a]. In abstract interpretation, the mathematical meaning of a program is derived in much the same way as with a normal semantics. However, this is not necessarily the standard meaning, but an approximation which can be used to extract information about the dynamic or run-time behaviour of the program. In essence, abstract interpretation is a mathematically justifiable approach that has the capability to statically analyse the dynamic properties of software applications at compilation time without executing the program itself. It is therefore clear that abstract interpretation attempts to fill-the-gap between conventional static analysis, that is static inspection of program code; and dynamic analyses, applying profiling over several possibly independent program executions. Another useful capability of an abstract interpretation is its ability to systematically analyse all possible paths of a program. It is because of these capabilities that abstract interpretation has found applications in a vast array of areas [NNH05]. The popularity of abstract interpretation has lead to the creation of commercial tools based on the analysis technique.

The natural ability of anytime computation in approximation algorithms lends itself to problems occurring in abstract interpretation in the presence of *Rice's theorem* [Ric53]. Rice's theorem states that, for any non-trivial property of a partial function, the question of whether a given algorithm computes a partial function with this property is formally undecidable. Rice's theorem is often referenced to show the undecidability of computing "interesting" properties of programs, and hence the undecidability of program analysis techniques in general. The force of this result is evident in that in order to even retain decidability, program analysis techniques are required to employ approximation techniques. Even the guarantee of decidability is often insufficient since in general decidability does not imply tractability. Hence, even in the presence of approximation techniques, a program analysis may still require considerable resources. The anytime computational paradigm can be seen to be especially important in this respect since anytime approximation algorithms allow for resource bounding. The potential impact of resource bounding is even more interesting if the coarseness of the

approximation attained is directly linked to the computational resource available (i.e. the algorithm is monotonic). If indeed the approximation algorithm utilised is monotonic, the anytime computation paradigm affords users the ability to attach resource limits to a program analysis directly affecting the accuracy of the analysis itself without compromising correctness.

The natural approach to controlling the cost of program analysis is to apply widening [CC92b] and one major contribution of this thesis is in providing an anytime algorithm for widening ROBDDs which were previously thought to be difficult to approximate [Mau98].

### Logic Synthesis

In engineering, anytime algorithms are particularly useful as many interesting problems are computationally hard [Qui52, Uma01]. Moreover, in practise, hard instances frequently arise [Bry96] so that computing an optimal answer is often infeasible. In such scenarios, an anytime algorithm is attractive and often matches the requirements of the engineering development cycle. For instance, consider the task of designing a logic circuit for future synthesis to silicon through some process. Assume the circuit itself is large enough for it to be computationally infeasible to verify the correctness of the entire circuit each time it is changed since this is likely to occur frequently. Thus it is prudent to possess some means to verify (to some restricted degree) the integrity of recent changes to the design. Such a technique permits us to reduce the possibility of the design possessing a flaw before production. However, before the circuit is realised in silicon, we may reapply the verification technique to further guarantee the integrity of the overall design by granting the verification process greater computational resources. The same requirements arise in software development, in particular software verification and automated discovery of bugs and security vulnerabilities.

Within logic synthesis, the anytime approach is not limited to verification tasks, in fact any algorithmic problem that frequently arises in the development cycle may benefit from anytime generality. One such problem is that of symmetry detection, which has applications in technology mapping [MD90, LSP92], combining technology-independent and technology-dependant stages of logic synthesis [KS00a], detecting support-reducing bound sets [ZCJMB05], ROBDD minimisation [PSP94, SMMD99], detecting equivalence of Boolean functions for which input correspondence is unknown [MM93, CM93a, ZCJMB04] and solving difficult instances of the Boolean satisfiability problem [ARMS02, DLSM04]. Another major contribution of this thesis is in providing an anytime algorithm for symmetry detection for Boolean formulae represented as ROBDDs. The state-of-the-art algorithm was based on a divide-and-conquer approach and therefore was inherently monolithic [Mis03].

## 1.2 Thesis Structure

**Chapter 2: Preliminaries and Background** In Chapter 2 we cover the required background and preliminaries relating to material used throughout this thesis. The first section covers the basics of *Boolean formulae*. We then proceed to introduce the concept of *Binary Decision Diagrams* along with other related representations. We conclude the chapter with a discussion of the complexity theoretic issues surrounding the use of the Reduced Ordered Binary Decision Diagram data structure, a popular extension to the Binary Decision Diagram where a variable order is enforced, and reduction rules apply.

**Chapter 3: Symmetry Detection in ROBDDs** In Chapter 3 we present an efficient, incremental, anytime algorithm for *first-order classical symmetry* detection. We explain how an incremental anytime approach offers special opportunities for optimisation, in that classical asymmetry/symmetry sieves, or fast linear time procedures to detect asymmetric variable pairs, can be applied before symmetry detection. Furthermore, asymmetry/symmetry propagation techniques can be inserted into the main loop of the algorithm. The practicality of our algorithm is then demonstrated through extensive experimental results.

**Chapter 4: Generalised Symmetry Detection in ROBDDs** In Chapter 4 we extend our anytime first-order classical symmetry detection algorithm for Boolean functions represented as ROBDDs to the *generalised* symmetry types over two variables. The extended algorithm infers all generalised two-variable symmetries, that is, all possible two-variable co-factor equivalences. The generalised algorithm retains the important attributes of the classical symmetry detection algorithm in that it is both anytime and efficient. The algorithm is underpinned by new implicational relationships between generalised symmetries, and these results are, in themselves, useful [ZMBCJ06].

**Chapter 5: Widening ROBDDs with Prime Implicants** In Chapter 5 we present a novel widening technique for Boolean formulae represented as ROBDDs. The chapter proposes a widening that can be used to both constrain the size of an ROBDD and also ensure that the number of times that it is weakened is bounded by some given constant. The widening can be used to either systematically approximate from above (i.e. derive a weaker function) or below (i.e. infer a stronger function).

**Chapter 6: Widening ROBDDs Randomly** In Chapter 6 we investigate the approximation of Boolean formulae represented as ROBDDs using randomisation techniques. These new approximation methods are capable of building ROBDD approximations without significant ROBDD manipulation, which is key to efficiency. Moreover, these algorithms remedy a deficiency in the widening based on prime implicants since

although the prime implicant widening is incremental, the increments themselves can be prohibitively expensive to compute. This chapter shows how the prime implicant approach can be refined to reduce the granularity of the increments and hence their cost, and thereby improve their anytime nature.

### 1.2.1 Publications

The research presented in this thesis has been published in the following papers:

[Chapter 3] N. Kettle and A. King. An Anytime Symmetry Detection Algorithm for ROBDDs. In Asia and South Pacific Design Automation Conference (ASPDAC), IEEE Press, pages 243 – 248, 2006. Acceptance rate: 30%.
**Contribution:** N. Kettle provided the initial algorithms, benchmarks and proofs.

[Chapter 4] N. Kettle and A. King. An Anytime Algorithm for Generalised Symmetry Detection in ROBDDs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), IEEE Press, April 2008.
**Contribution:** N. Kettle provided the initial algorithms, benchmarks and proofs.

[Chapter 5] N. Kettle, A. King and T. Strzemecki. Widening ROBDDs with Prime Implicants. In Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Springer, pages 105 – 119, 2006. Acceptance rate: 22%.
**Contribution:** N. Kettle provided an *ROBDD* based algorithm for computing prime implicants of a given length, benchmarks and proofs. T. Strzemecki provided a detailed algorithm for computing prime implicants of a given length in Boolean functions for whom a truth table is available.

# Chapter 2

# Background and Preliminaries

"Beginning is easy - continuing hard."
- Japanese Proverb

**Abstract.** In this chapter we introduce the necessary preliminaries relating to material used throughout this thesis. This chapter covers the basics of Boolean formulae, and the concept of a binary decision diagram.

## 2.1 Boolean Functions

Boolean logic is named after Boole, an English mathematician who was first to define an algebraic system of logic based on 0s and 1s. Shannon expanded upon the theories of Boole by showing how electric circuits utilising relays were a model for Boolean logic. This result was to prove enormously important with the emergence of the electronic computer, and the development of computer science as we understand it today. For brevity we shall refrain from giving a detailed discussion of the origins of Boolean logic, since this is beyond the scope of this thesis, and instead proceed to define the necessary foundations for the concepts used throughout this thesis.

**Definition 2.1** (Boolean Function)**.** *A Boolean function on $n$ variables is a function $f : \mathbb{B}^n \to \mathbb{B}$, where $\mathbb{B}$ is the set $\{0, 1\}$, $n$ is a positive integer, and $\mathbb{B}^n$ denotes the $n$-fold cartesian product of set $\mathbb{B}$ with itself.*

A Boolean function $f : \mathbb{B}^n \to \mathbb{B}$ can be represented by a propositional formula over an ordered set of variables $X$ where $|X| = n$.

**Example 2.1.** *The Boolean functions $f, f' : \mathbb{B}^2 \to \mathbb{B}$ where $f = \{\langle 0,0\rangle \mapsto 0, \langle 0,1\rangle \mapsto 1, \langle 1,0\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1\}$ and $f' = \{\langle 0,0\rangle \mapsto 0, \langle 0,1\rangle \mapsto 0, \langle 1,0\rangle \mapsto 0, \langle 1,1\rangle \mapsto 1\}$ can be represented as $f = (x_1 \vee x_2)$ and $f' = (x_1 \wedge x_2)$.*

The set of Boolean formulae, or propositional formulae, over a finite set of variables $X$ is denoted $\mathbb{B}_X$ and henceforth functions and formulae will be used interchangeably.

### 2.1.1 Representations and Operations

One commonly used representation of a Boolean function $f$ is its truth table, that is, to give a complete list of all binary vectors $\mathbf{b} \in \mathbb{B}^n$ under which the function evaluates to *true*. Formally, we define the set of all binary vectors under which a Boolean function evaluates to *true* to be its model set.

**Definition 2.2** (Model Set)**.** *The set of models of a Boolean function $f \in \mathbb{B}^n \to \mathbb{B}$ over a set of variables $X$ is defined as $model_X(f) = \{\langle b_1, \ldots, b_n\rangle \mid f(b_1, \ldots, b_n) = 1\}$.*

**Example 2.2.** *Given $X = \{x_1, x_2, x_3\}$ and $f = x_1 \wedge (x_2 \to x_3)$ then $model_X(f) = \{\langle 1,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,1\rangle\}$.*

Furthermore, we define the *satisfy-count* (*minterms*) of a Boolean function $f$ as $\|f\| = |model_X(f)|$.

**Definition 2.3** (Entailment)**.** *A Boolean function $f \in \mathbb{B}_X$ is said to entail another $f' \in \mathbb{B}_X$, denoted $f \models f'$ if $model_X(f) \subseteq model_X(f')$.*

**Example 2.3.** *Given two Boolean functions $f, f' \in \mathbb{B}_X$ such that $f = x_1 \wedge (x_2 \to x_3)$ and $f' = (x_1 \wedge \neg x_2 \wedge \neg x_3)$ where $X = \{x_1, x_2, x_3\}$ then $f'$ entails $f$, that is, $f' \models f$ since $model_X(f') = \{\langle 1,0,0\rangle\} \subseteq \{\langle 1,0,0\rangle, \langle 1,0,1\rangle, \langle 1,1,1\rangle\} = model_X(f)$.*

**Definition 2.4** (Equivalence)**.** *A Boolean function $f \in \mathbb{B}_X$ is said to be equivalent to another $f' \in \mathbb{B}_X$, denoted $f \equiv f'$ if and only if $model_X(f) = model_X(f')$, or equivalently $f \models f' \wedge f' \models f$.*

The set of all propositional formulae over a set of propositional variables $X$ forms a complete lattice $\langle \mathbb{B}_X, \models, 0, 1, \vee, \wedge, \neg\rangle$ where $\models$ denotes entailment (Definition 2.3), $\wedge$-logical and, $\vee$-logical or, $\neg$-negation and 0 (*false*), 1 (*true*) abbreviate the Boolean functions $\lambda\mathbf{b}.0$ and $\lambda\mathbf{b}.1$ respectively for all binary vectors $\mathbf{b} \in \mathbb{B}^n$.

**Definition 2.5** (Chains and Anti-Chains)**.** *We define a chain of Boolean functions $C$ to be a set $C \subseteq \mathbb{B}_X$ such that either $f \models f'$ or $f' \models f$ for all $f, f' \in C$. Conversely, we define an anti-chain of Boolean functions $A$ to be a set $A \subseteq \mathbb{B}_X$ such that $f \not\models f'$ or $f = f'$ for all $f, f' \in A$.*

**Definition 2.6** (Shannon Co-Factor)**.** *The Shannon co-factor of a Boolean function $f \in \mathbb{B}_X$ w.r.t. a variable $x_i \in X$ and a Boolean constant $b \in \mathbb{B}$ is defined by,*

$$f|_{x_i \leftarrow b} = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$$

*Multiple variable co-factors w.r.t. an ordered set of variables $X' \subseteq X$ such that $m = |X'|$ and a binary vector $\mathbf{b} \in \mathbb{B}^m$ denoted $f|_{x'_1 \leftarrow b_1, \ldots, x'_m \leftarrow b_m}$, are defined as $f|_{x'_1 \leftarrow b_1, \ldots, x'_m \leftarrow b_m} = f_m$ where $f_0 = f$ and $f_j = f_{j-1}|_{x'_j \leftarrow b_j}$.*

**Definition 2.7** (Shannon Expansion)**.** *The Shannon expansion of a Boolean function $f \in \mathbb{B}_X$ w.r.t. a variable $x_i \in X$ is the identity,*

$$f = (\neg x_i \wedge f|_{x_i \leftarrow 0}) \vee (x_i \wedge f|_{x_i \leftarrow 1})$$

*In this context, $f|_{x_i \leftarrow 1}$ and $f|_{x_i \leftarrow 0}$ are the positive and negative Shannon co-factors of the Boolean function $f$.*

The work of Shannon on circuit and communications theory, in particular the Shannon expansion theorem, paved the way for the construction of many computational techniques to tackle problems relevant to Boolean formulae. The importance of the Shannon expansion theorem will become apparent in §2.2.

**Definition 2.8** (Existential Quantification)**.** *The existential quantification of a Boolean function $f \in \mathbb{B}_X$ w.r.t. a variable $x_i \in X$ denoted $\exists_{x_i}(f)$ is defined as Schröder elimination, that is, $\exists_{x_i}(f) = f|_{x_i \leftarrow 0} \vee f|_{x_i \leftarrow 1}$.*

### 2.1.2 Decompositions

**Definition 2.9** (Cube)**.** *A cube $p$ is a Boolean function of the form $(\wedge_{y \in Y} y) \wedge (\wedge_{z \in Z} \neg z)$ such that $Y \cup Z \subseteq X$ and $Y \cap Z = \emptyset$ where $Y, Z$ are disjoint sets of variables; moreover, the length of $p$ is denoted $|p|$ and defined by $|p| = |Y| + |Z|$.*

**Definition 2.10** (Prime Implicant)**.** *A prime implicant $p$ of a Boolean function $f \in \mathbb{B}_X$ is a cube $p$ such that $p \models f$, i.e. $p$ is an implicant of $f$; and there exists no other implicant $p'$ of $f$ such that $p \models p'$ and $p' \neq p$.*

For brevity, we let $primes(f)$ denote the set of all prime implicants of the Boolean function $f$. Finally, observe $primes(true) = \{true\}$ and $primes(false) = \emptyset$.

**Example 2.4.** *Let $X = \{x_1, x_2, x_3, x_4\}$ and $f = (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_3 \vee x_4)$ and $p = (\neg x_1 \wedge \neg x_3)$. Observe that $p \models f$ and therefore $p$ is a implicant of $f$. Further, suppose $p \models p'$ and $p \neq p'$. Then $p' = \neg x_1$ or $p' = \neg x_3$ and, in either case, $p' \not\models f$. Hence $p$ is a prime implicant of $f$. In fact, $primes(f) = \{\neg x_1 \wedge \neg x_3, \neg x_2 \wedge \neg x_3, \neg x_1 \wedge x_4\}$.*

## 2.2 Binary Decision Diagrams

The *Binary Decision Diagram* (BDD) is a widely-known, efficient tree-based data structure for the representation of Boolean logic functions [Ake78, Bry86, Bry92] (in fact Bryant's paper [Bry86] is one of the most highly cited papers in computer science). The BDD is most well-known for its broad applications in model checking [CGP00], a method for algorithmically verifying a formal system by checking if a model, often constructed from a hardware or software design, satisfies a given formal specification. The ability of

BDDs to compactly represent large state-spaces in sub-exponential space has allowed massive improvements in the efficiency of such model checkers [BCM+92]. BDDs have also seen many applications in test generation [ABA95], fault simulation [MWBSV88, Bry96], checking Boolean satisfiability [ADG91], program analysis [WL04], constraint solving [HLS04, HLS05] and abstract interpretation [Fec97, BS99, LS04].

**Definition 2.11** (BDD)**.** *A BDD is a rooted directed acyclic graph where each internal node is labeled with a Boolean variable $x_i$. Each internal node has one successor node connected via an edge labeled* 0*, and another successor connected via an edge labeled* 1*. Each leaf node is either the Boolean constant* 0 *or* 1*.*

The Boolean function represented by a BDD can be evaluated for a given variable assignment, that is, a binary vector $\mathbf{b} \in \mathbb{B}^n$; by traversing the graph from the root, taking the 1 edge at a node when the variable is assigned to 1 and the 0 edge when the variable is assigned to 0. The leaf node reached in this traversal indicates the value of the Boolean function for the variable assignment. Observe that each sub-BDD of a BDD also itself represents a Boolean function. For brevity, we let $|g|$ denote the number of internal nodes in the BDD $g$, the function $index(g)$ the variable with which the BDD node $g$ is labelled and the function $var(g)$ the variable set over which the ROBDD $g$ is defined.

The concept of the BDD is enriched with an ordering so as to improve the efficiency of BDD operations [Bry86]. This BDD variant is known as the *Ordered-Binary Decision Diagram* (OBDD).

**Definition 2.12** (OBDD)**.** *An OBDD is obtained from a BDD by imposing a total ordering on the variables over which the BDD is defined and applying the restriction that the label of a node is always less than the label of any internal node in its two successors.*

The concept of a *Reduced-Ordered Binary Decision Diagram* (ROBDD) [Bry86] is derived from an OBDD so as to obtain a canonical representation, in which equivalent Boolean functions are represented by the same graph.

**Definition 2.13** (ROBDD)**.** *An ROBDD is an OBDD obeying the following restrictions to obtain an efficient reduced representation:*

- *There can exist no sub-ROBDD that is rooted at a node labelled with a variable $x_i$ that represents a function $f$ such that $f|_{x_i \leftarrow 0} = f|_{x_i \leftarrow 1}$.*
- *There are no two nodes labelled with the same variable that have identical successor nodes.*

An example ROBDD for the function $f = (x_1 \wedge x_2)$ can be found in Figure 4.

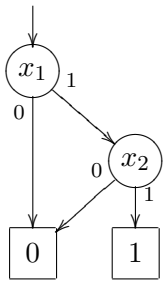


Figure 4: ROBDD of $f = (x_1 \wedge x_2)$

### 2.2.1 Variable Order and Minimisation

ROBDDs represent Boolean functions by placing a total order on their variables. Any reordering of these variables may yield differing ROBDDs for the same Boolean function of differing size. An inappropriate variable ordering may result in the construction of an ROBDD of exponential size, even if the represented function is relatively simple. Hence, the variable ordering problem — the problem of computing a variable ordering that results in a reduction in ROBDD size — is considered fundamental to the efficient manipulation of ROBDDs. To illustrate the importance of variable ordering, consider the Boolean function $f = \bigvee_{i=1}^{n}(x_i \wedge y_i)$, which is the so-called Achilles heel function [Sas99, BSM05]. The resultant ROBDD can be shown to grow linearly in the number of variables $n$ for the interleaved variable ordering $x_1 \prec y_1 \prec x_2 \prec y_2 \prec \cdots \prec x_n \prec y_n$, but has size $O(2^{\frac{n}{2}+1} - 2)$ [Sas99] for the disjoint variable ordering $x_1 \prec x_2 \prec \cdots \prec x_n \prec y_1 \prec y_2 \prec \cdots \prec y_n$. The ROBDDs for the Achilles heal function under each of these variable orderings is given in Figure 5; in Figure 5, labels associated with nodes correspond to unique identifiers whilst dashed-edges correspond to *complement* edges [BRB90], that is, edges that have the effect of negating the eventual result of a variable assignment.

Given the ability of ROBDDs to concisely represent a large number of Boolean functions, the study of ROBDD space complexity for various types of Boolean formulae has attracted much interest. Results of Bryant [Bry91] provide exponential lower bounds on the space complexity on several functions such as the middle bit of the ROBDD representation of an $n$-bit multiplier [Bry91]. The complexity orders of integer addition and multiplication when represented with an ROBDD have respectively $\Omega(n), O(2^n)$ and $\Omega(2^n), O(2^n)$ best and worst-case complexities [Bry91, Min96]. Interestingly, as Wegener observed, the choice of a good variable ordering for some functions, i.e. those of addition and bitwise equality test, is essential since their ROBDD representation is of exponential size under almost all variable order permutations [Weg00]. Hence efficient algorithms to obtain a suitable variable ordering are essential in many applications.

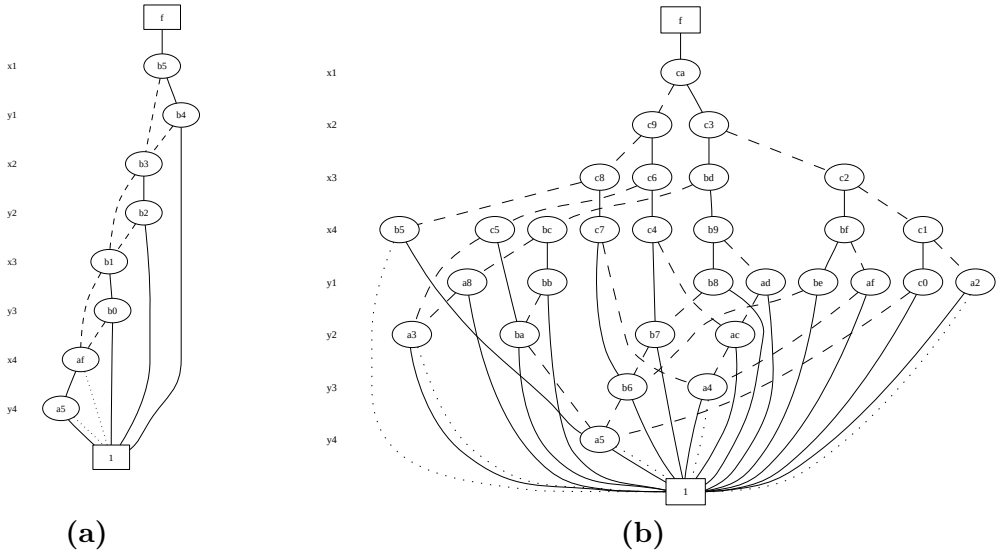


Figure 5: ROBDD of $f = \bigvee_{i=1}^{n}(x_i \wedge y_i)$ where $n = 4$ with (a) optimal linear variable order and (b) exponential variable order.

### Optimal Variable Reordering

Currently, the best known algorithm to compute the optimal variable ordering of a Boolean function represented as an ROBDD is the dynamic programming approach of Friedman and Supowit [FS90]. The algorithm resides in $O(n^2\, 3^n)$ where $n$ is the number of variables in the ROBDD. The algorithm requires the truth-table of the Boolean function to be enumerated and is therefore considered to be impractical for anything but small $n$ since the space complexity is in $\Omega(2^n)$. (However, their result is a significant improvement over the previous best known algorithm of Nair and Brand [NB86] which required $O(n!\, 2^n)$ time.) The inability of researchers to find better algorithms for the optimal variable ordering problem led to the assumption that the problem is computationally hard [BW96].

### Heuristic Variable Reordering

For lack of an optimal deterministic polynomial time algorithm to solve the optimal variable ordering problem, a large number of heuristic approaches have been suggested. These heuristics are based upon either *local* [Rud93, FYBSV93] (switching of adjacent variables) or *global* [NB86, FS90] changes (switching of arbitrary variables, or the construction of an entirely new variable ordering). The most popular of these heuristics is based upon so-called *sifting* introduced by Rudell [Rud93]. Sifting is based on the

observation that two variables adjacent in the order can be switched efficiently. In fact the cost of such a swap is merely linear in the number of nodes labelled with the swapped variables. A variable is then selected, and shifted through the variable ordering, searching for a propitious position for the variable. Although swapping two adjacent variables is inexpensive, the total time for sifting grows rapidly with the number of variables [MS97].

Computing a propitious variable ordering is not only useful for the minimisation of ROBDDs. The efficiency of an ROBDD representation of a Boolean function may also be measured in terms of the Average Path Length (APL) of the ROBDD [BSM05], that is, the mean number of decisions required to evaluate an assignment of the variables [EGD04]. Possessing a low APL is particularly important for applications in logic synthesis which require repeated and extensive ROBDD evaluation. Interestingly, the variable ordering required to minimise the APL does not correspond to the variable ordering required to minimise the number of nodes [NMSB03]. Thus minimisation of the APL of an ROBDD does not constitute a useful heuristic for ROBDD minimisation.

#### Complexity of Variable Reordering

Bollig and Wegener [BW96] later settled the complexity question of the optimal variable ordering problem by proving the following problem NP-complete. Given an ROBDD $g$ of a Boolean function $f$, decide if there exists an ROBDD $g'$ representing $f$ with at most $s$ nodes [BW96]. Hence, the problem of computing an optimal variable ordering is NP-hard [Weg00]. Furthermore, to make matters worse, not only is the optimal variable ordering problem NP-complete, but Sieling [Sie02] recently proved the problem inapproximable within a constant factor. That is, if for every given $\epsilon > 0$, there exists a polynomial-time algorithm for reordering variables so as to obtain an ROBDD whose size is not larger than $1+\epsilon$ times that of the minimal size, then it follows that P = NP. Hence, it is impossible to devise a polynomial time approximation algorithm to approximate the optimal variable ordering unless P = NP. Thus, research has continued with the development of further heuristic based approaches, however, the results of Sieling [Sie02] severely limit the effectiveness of such algorithms as they cannot guarantee any useful improvement whilst retaining polynomial run time.

### 2.2.2 The Complexity of ROBDD Operations

The ability of ROBDDs to canonically and concisely represent Boolean formulae, implies many interesting complexity theoretic properties. Table 1 summarises the complexities of common ROBDD operations as defined by Bryant [Bry86]. The purpose of each of these is detailed below.

- `Reduce` - derive the canonical ROBDD representation of an OBDD $G$.

- `Apply` - this can be instantiated two ways, to either compute the logical-and, or the logical-or of two Boolean functions $f_1, f_2 \in \mathbb{B}_X$ represented as ROBDDs $G_1$ and $G_2$ respectively. Specifically an OBDD $G$ is computed that represents the Boolean function $f_1 \cdot f_2$ for some operation $\cdot \in \{\wedge, \vee\}$. Note, this operation is traditionally followed by a `Reduce` step.

- `Restrict` - for some $f \in \mathbb{B}_X$ represented as an ROBDD $G$, compute an OBDD $G'$ that represents the Shannon Co-Factor $f|_{x_i \leftarrow b}$ of $f$ w.r.t. the variable $x_i \in X$ and Boolean constant $b \in \mathbb{B}$. Note again that this operation is followed by a `Reduce` step.

- `Negate` - for some $f \in \mathbb{B}_X$ compute the function $\neg f$. The complexity for `Negate` in Table 1 is $O(1)$ for BDD packages supporting complement-edges, and is $O(|G|)$ for packages that do not implement this refinement. Note that `Reduce` is not needed in either case after the `Negate` step.

- `Satisfy-one` - this is a fundamental operation that must be supported by any representation of a Boolean function. Specifically, for some $f \in \mathbb{B}_X$, compute $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$, that is, compute a binary vector $\mathbf{b}$ under which the Boolean function $f$ evaluates to *true*. This operation is trivial for an ROBDD representation.

- `Satisfy-all` - this too is another fundamental Boolean function operation. The operation computes the truth-table of a Boolean function $f$. Specifically, for some $f \in \mathbb{B}_X$, compute every $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$. Again, this operation is straightforward for an ROBDD representation though not necessarily tractable.

- `Satisfy-count` - Intuitively, this operation computes the number of truth assignments for a Boolean function $f$. To be precise, for some $f \in \mathbb{B}_X$, this operation computes $|\mathit{model}_X(f)|$, that is, the satisfy-count of the Boolean function $f$.

Finally, the remainder problems `SAT`, `TAUT` and `EQUIV` correspond to the classic problems of Boolean *satisfiability* [Coo71], *tautology* [Wei] and Boolean function *equivalence* respectively. The table gives the best known complexities for the operations above rather than the complexities originally reported by Bryant [Bry86]. The complexity given for the `Reduce` operation is that reported by Sieling and Wegener [SW93] who showed that bucket sort may be used to reduce the complexity bound from $O(|G| \log |G|)$ to $O(|G|)$. This refinement has no impact on the complexities of the other operations, however, `Reduce` is reapplied after the `Apply` and `Restrict` operations.

The complexity of many ROBDD operations is polynomial in the size of the input. The exception to this is the `Satisfy-all` operation, that outputs all satisfying assignments for a given ROBDD. Note that the `Satisfy-count` problem, that is, the

| Operation | Complexity |
|---|---|
| `Reduce` | $O(\|G\|)$ |
| `Apply` | $O(\|G_1\| \cdot \|G_2\|)$ |
| `Restrict` | $O(\|G\| \cdot \log \|G\|)$ |
| `Negate` | $O(1)$ |
| `Satisfy-one` | $O(n)$ |
| `Satisfy-all` | $O(n \cdot \|model_X(f)\|)$ |
| `Satisfy-count` | $O(\|G\|)$ |
| `SAT` | $O(1)$ |
| `TAUT` | $O(1)$ |
| `EQUIV` | $O(1)$ |

Table 1: ROBDD operation complexities [Bry86, SW93]

problem of computing the total number of satisfying assignments, resides in $O(|G|)$ for an ROBDD $G$. Interestingly, the problem of computing the number of satisfying assignments for a Boolean function is believed to be harder than even NP itself when the function is presented in `CNF` [GJ79].

# Chapter 3

# Symmetry Detection in ROBDDs

"Symmetry is a complexity-reducing concept;
seek it everywhere."
- A. Perlis

**Abstract.** In this chapter we present an anytime algorithm for *classical* symmetry detection for Boolean functions represented as ROBDDs. The algorithm seeks to address some of the drawbacks associated with existing symmetry detection methods that have been proposed for ROBDDs.

## 3.1 Introduction

Symmetry detection has been important since the days of Shannon [Sha38] who observed that symmetric functions have particularly efficient switch network implementations. Symmetry detection is no less important these days and knowledge of symmetric variables has found many applications in logic synthesis [KD91, EH78], technology mapping [MD90, LSP92], combining technology-independent and technology-dependant stages of logic synthesis [KS00a], detecting support-reducing bound sets [ZCJMB05], ROBDD minimisation [PSP94, SMMD99], detecting equivalence of Boolean functions for which input correspondence is unknown [MM93, CM93a, ZCJMB04] and solving difficult instances of the Boolean satisfiability problem [ARMS02, DLSM04].

The challenge in *classical* symmetry detection is to find efficient algorithms for detecting all *classically* symmetric variables pairs $(x_i, x_j)$ of a given Boolean function $f \in \mathbb{B}_X$ over a set of variables $X$. A Boolean function $f \in \mathbb{B}_X$ is classically symmetric in a pair of variables $(x_i, x_j) \in X^2$ if and only if $f(x_0, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_0, \ldots, x_j, \ldots, x_i, \ldots, x_n)$. This notion of symmetry implies that $f$ remains unchanged under the switching of the variables $x_i$ and $x_j$. This symmetry property can be restated as a co-factor equivalence by using the following observation: if $f$ is classically symmetric in $(x_i, x_j)$ then it follows that $f(x_0, \ldots, 0, \ldots, 1, \ldots, x_n) = f(x_0, \ldots, 1, \ldots, 0, \ldots, x_n)$

hence $f|_{x_i \leftarrow 0, x_j \leftarrow 1} = f|_{x_i \leftarrow 1, x_j \leftarrow 0}$. Furthermore, if $f|_{x_i \leftarrow 0, x_j \leftarrow 1} = f|_{x_i \leftarrow 1, x_j \leftarrow 0}$ then $f(x_0, \ldots, 0, \ldots, 1, \ldots, x_n) = f(x_0, \ldots, 1, \ldots, 0, \ldots, x_n)$. Moreover, it vacuously follows that $f(x_0, \ldots, x_i, \ldots, x_j, \ldots, x_n) = f(x_0, \ldots, x_j, \ldots, x_i, \ldots, x_n)$ whenever $x_i = x_j$. Hence $(x_i, x_j)$ is classically symmetric if and only if $f|_{x_i \leftarrow 0, x_j \leftarrow 1} = f|_{x_i \leftarrow 1, x_j \leftarrow 0}$ [HS96]. To differentiate classically symmetric variable pairs $(x_i, x_j)$ from the generalised symmetry types introduced later in this thesis, we say a pair of variables $(x_i, x_j)$ is $T_1$-symmetric in a Boolean function $f \in \mathbb{B}_X$ whenever the pair is classically symmetric in the Boolean function $f$. This symmetry is formally known as the first-order classical symmetry, or the non-skew non-equivalence symmetry [KS00b].

The algorithms used to compute the symmetries of a Boolean function critically depend on the representation of the Boolean function. The most well-known of these representations, `CNF`, is used primarily because of the straightforwardness of deriving `CNF` [Tse68], furthermore, it is the standard input representation for a `SAT` solver. Algorithms for computing symmetries in Boolean formulae represented as `CNF` [DLSM04,ZMBCJ06] utilise a mapping between the `CNF` instance of a Boolean function $f$ and a graph $G$ such that a symmetry in $f$ corresponds to a non-trivial graph *automorphism* in $G$, that is a non-trivial mapping of the vertices of $G$ to $G$ such that the resulting graph is isomorphic with $G$. Since the presence of a symmetry implies that the graph has a non-trivial automorphism, it follows that the problem of finding a symmetry is at least as hard as graph automorphism (`GA`) [KST93]. Although the `GA` problem is considered to be outside of P (`GA` $\notin$ P), it is believed to not be NP-complete and hence resides in a complexity class $k$ between P and NP [KST93] (P $\subseteq k \subseteq$ NP). Contrary to what one would expect, finding `CNF` symmetries by such means has been shown, albeit experimentally, to be easier than solving `SAT` [ARMS02]. Furthermore, excellent graph automorphism algorithms are known and are typically capable of returning a small set of irredundant automorphism generators [McK81]. However, the complexity of `GA` remains unknown, and thus so too is the complexity of symmetry finding (at least in the case of `CNF` representations).

## 3.2 Applications

ROBDDs revolutionised logic synthesis [Bry86], and model checking [BCM+92] since they offer a compact representation for many Boolean functions, and hence there has been much interest in extracting symmetries from an ROBDD representation. Even when a Boolean function can be represented efficiently as an ROBDD, state-of-the-art symmetry computation algorithms can take an exorbitant amount of time [KK06]. In an effort to eleviate this computational problem, this chapter presents an efficient anytime algorithm for classical symmetry detection for Boolean functions represented as ROBDDs.

Symmetry detection has many applications in problems contained within the process of logic synthesis [KD91, EH78, MD90, LSP92, KS00a, ZCJMB05, PSP94, SMMD99, MM93, CM93a, ZCJMB04, ARMS02, DLSM04]. Notable examples include ROBDD minimisation and Permutation independent Boolean comparison, these are discussed in the following sections followed by a discussion of current algorithms.

### 3.2.1 ROBDD minimisation

Several ROBDD minimisation techniques have been proposed that utilise symmetry information pertaining to the underlying Boolean function [PSP94, SMMD99]. These techniques are justified by the observation that symmetric variables tend to be adjacent in optimal variable orderings [SMMD99, HLM00]. This has been shown experimentally for all Boolean functions with up to five variables [HLM00]. Further, it can be shown that the size of an ROBDD representing a totally symmetric Boolean function, that is, a Boolean function which remains invariant under all variable permutations, is bounded by $O(|X|^2)$ [SMMD99]. More refined bounds have been derived for ROBDDs that possess only a small number of symmetries [HLM00]. These bounds limit the size of an ROBDD that is obtained under the variable ordering which places symmetric variables contiguously. The bounds substantiate the practise of placing symmetric variables adjacent in the variable ordering.

### 3.2.2 Permutation Independent Boolean Comparison

Verifying the equivalence of two Boolean functions $f, f' \in \mathbb{B}_X$ is a constant time operation (or linear, in the worst-case) when the ROBDD representations of the functions $f$ and $f'$ are given. However, simply comparing the canonical forms of $f$ and $f'$ is not sufficient to verify equivalence if the correspondence between the variables over which $f$ and $f'$ operate is unknown [MM93, CM93a]. For instance, consider verifying the equivalence of two Boolean functions $f \in \mathbb{B}_X$ and $f' \in \mathbb{B}_{X'}$ such that $X \cap X' = \emptyset$. A solution to this problem constitutes a mapping $\rho : X \to X'$ such that,

$$f \equiv \exists_{\rho(x_1)} \ldots \exists_{\rho(x_n)} \left( f' \wedge \bigwedge_{x_i \in X} (x_i \iff \rho(x_i)) \right)$$

The application of classical symmetries in permutation independent Boolean comparison is then clear, that is, if the functions $f$ and $f'$ are indeed equal, then both functions must possess an isomorphic set of symmetry pairs. That is, if $(x_i, x_j)$ is a symmetry pair of function $f$ then $(\rho(x_i), \rho(x_j))$ is a symmetry pair of $f'$ and vice versa. The force of this observation is that it places structural constraints on $\rho$, that is, $\rho(x_i) \neq x'_j$ if the number of variables that are symmetric with $x_i$ in $f$ differs from the number of variables symmetric with $x'_j$ in $f'$ [ZCJMB04].

## 3.3 Problems with Existing Methods

In problems relating to logic synthesis it is often more attractive to find an acceptable answer in a reasonable amount of time rather than the optimal answer in an exorbitant amount of time. For instance, in [ZCJMB05] it is remarked that a heuristic approach to support reducing bound set computation can detect over 80% of all support-reducing bound sets, while achieving a 40 fold runtime reduction compared to the exhaustive method [ZCJMB05]. Such heuristic methods are acceptable in such a scenario, however, the solution obtained is not guaranteed to be complete. In classical symmetry detection, the requirements of an algorithm are much the same, however experimental results have indicated that the running time of state-of-the-art algorithms [Mis03, ZCJMB04] can exceed 12 hours on some ROBDDs of less than a million nodes (the problem is further exacerbated since ROBDDs of a million nodes, or greater, in size is an increasingly common occurrence [FGH$^+$05]).

Variable reordering can reduce the size of an ROBDD and thereby reduce the cost of symmetry detection by current state-of-the-art methods. However, it is imprudent to rely on variable reordering alone to make classical symmetry detection tractable since variable reordering techniques can themselves be prohibitively expensive and of course, even after reordering, there is no guarantee that the size of the ROBDD will actually be smaller. In fact, as we remarked in §2.2.1, even merely improving the variable ordering is NP-complete [BW96], and is also inapproximable within a constant factor [Sie02]. From the perspective of algorithm design, there are at least two ways forward: develop a faster symmetry detection algorithm; recast symmetry detection so that it can be solved with an anytime algorithm. Anytime algorithms arise in engineering tasks when it is more attractive to find an acceptable answer in a reasonable amount of time rather than the optimal answer in an exorbitant amount of time. In the context of classical symmetry detection the challenge is therefore to devise an efficient algorithm that incrementally detects pairs of $T_1$-symmetric variables until some given time bound is exceeded. Thus far, the only incremental algorithms that have been proposed for symmetry detection are those based on naïve co-factor computation [MMW93, SMMD99], but alas, this approach is inefficient. The algorithm of Panda *et al.* [PSP94] can be considered to be incremental and it does not require co-factor computation, but the algorithm is incomplete for the purposes of symmetry detection, that is, the algorithm cannot be relied upon to compute all $T_1$-symmetric variable pairs of a given Boolean function.

## 3.4 Contributions

In this chapter we present an efficient anytime algorithm for classical symmetry detection. For clarity, we summarise our contributions as follows:

- We present an incremental, anytime algorithm for first-order classical symmetry detection for Boolean functions represented as ROBDDs. Even considering the complexity of all the underlying set operations, the algorithm is in $O(n^3 + n|G| + |G|^3)$ where $n$ is the number of variables and $|G|$ the number of nodes in the ROBDD. Since $n \ll |G|$, it follows that the algorithm is competitive with Mishchenko's algorithm.

- We explain how an incremental anytime approach offers special opportunities for optimisation, in that classical assymetry/symmetry sieves can precede the algorithm and assymetry/symmetry propagation techniques can be inserted into the main loop of the algorithm.

- Further, we propose a computationally lightweight technique that often improves the proportion of symmetries found early on in the operation of the algorithm and hence further increases the diminishment of intermediary results.

The remainder of this chapter is structured thus, Section 3.5 surveys the related work. Section 3.6 presents an anytime symmetry detection algorithm for classical symmetries. Due to the way the anytime algorithm decomposes symmetry detection into a series of passes, one for each variable, we are free to exploit transitivity relations in order to propogate asymmetry/symmetry information before each of these passes to reduce the expected cost of each pass. Furthermore, before the first pass, it is possible to use preprocessing algorithms — sieves — that detect pairs of asymmetric variables and thereby further reduce the number of variable pairs that need to be considered and hence decrease the cost of symmetry detection. Section 3.7 and 3.8 explain and quantify the value of these refinements. Finally, Section 3.9 presents the concluding discussion.

## 3.5 Related Work

Early work on detecting symmetric variables in Boolean functions has focussed on the computation of co-factor pairs, that is all $n^2 - n$ possible co-factors, where $n$ is the number of variables. Symmetry is detected by checking their equivalence [MMW93]. The use of ROBDDs to represent Boolean functions enables not only the efficient computation of co-factors, but also equivalence to be checked in constant time. However, repeated co-factoring involves the creation and deletion of many intermediate ROBDD nodes and for very large ROBDDs this overhead can be prohibitive. This method is often referred to as the naïve method [MMW93]. Möller, Mohnke and Weber [MMW93] thus advocate the use of preprocessing algorithms — sieves — that detect pairs of asymmetric variables. These linear-time sieves significantly reduce the number of co-factor pairs that need to be computed. In general, however, methods built upon such sieves still require

naïve co-factor computation, that is, calls to the standard co-factoring algorithm the complexity of which is in $O(|G| \log |G|)$ [Bry86].

Because of the cost of repeated co-factoring, many symmetry detection methods endeavour to avoid naïve co-factor computation. Möller *et al.* [MMW93] and Panda *et al.* [PSP94] detect all symmetries between variables adjacent in the variable order with an algorithm in $O(|G|)$. Panda *et al.* [PSP94] modify Rudell's dynamic variable reordering algorithm [Rud93] to detect symmetries between variables that become adjacent when one of the variables is repositioned in the ROBDD variable ordering. Symmetric variables are then grouped, and any subsequent reordering that is applied is required to preserve a contiguous variable ordering within each group. This approach to symmetry detection does not require naïve co-factor computation, but there is no guarantee that all symmetries will be found.

Current state-of-the-art in classical symmetry detection is represented by the algorithm of Mishchenko [Mis03] that detects all symmetric variable pairs in a ROBDD with $O(|G|^3)$ set operations. Symmetric pairs are represented as sets which are stored as zero suppressed binary decision diagrams (ZBDDs [Min93]) which offer a compact representation of a collection of sets, in this case a collection of symmetric variable pairs. Since each set can potentially contain $O(n^2)$ elements one would expect Mishchenko's algorithm to at least reside in $O(n^2|G|^3)$ and possibly even a higher complexity class when all set operations are considered. Furthermore, these algorithms are monolithic in that they provide no opportunity for early termination, and can be shown to require prohibitive run time.

## 3.6 Anytime Symmetry Detection Algorithm

In this section we describe our anytime approach to classical symmetry detection. For pedagogical purposes we first present Algorithm 1 which is our simplest algorithm for anytime symmetry detection. In the section that follows, we build on Algorithm 1 by incorporating optimisations that exploit its anytime nature. Algorithm 1 takes as input an ROBDD $f$ and returns a set of index pairs $S = \{(i,j) \mid {T_1}^{x_i,x_j}(f)\}$ where the predicate ${T_1}^{x_i,x_j}(f)$ indicates that $f$ is classically symmetric in the variable pair $(x_i, x_j)$. The algorithm is composed of two separate procedures: `FindAsymmetry` and `RemoveAsymmetry`. `FindAsymmetry`$(f)$ performs two depth-first traversals over the ROBDD $f$ to detect pairs of variables $(x_i, x_j)$ that are provably asymmetric with respect to $T_1$. `RemoveAsymmetry`$(f, i, C)$ filters a set of variable indices $C$ whose symmetry relationship with variable $x_i$ is unknown to return the set $C' \subseteq C$ that represents those variables $x_j$ that are $T_1$-symmetric with $x_i$.

The call to `FindAsymmetry` initialises the set of $T_1$-asymmetric variable pairs $A$ such that $A \subseteq \{(i,j) \mid \neg {T_1}^{x_i,x_j}(f)\}$ after which the set $S$ of $T_1$-symmetric variable

**Algorithm 1** SymmetricPairs($f$)

**Require:** $f \in ROBDD_X$
1: $A \leftarrow$ `FindAsymmetry(f)`
2: $S \leftarrow \emptyset$
3: **for** $i = 1$ **to** $n - 1$ **do**
4: $\quad C \leftarrow \{\, j \mid (i,j) \notin (A \cup S) \wedge i < j\}$
5: $\quad D \leftarrow$ `RemoveAsymmetry`$(f, i, C)$
6: $\quad A \leftarrow A \cup \{(i,l),(l,i) \mid l \in C \setminus D\}$
7: $\quad S \leftarrow S \cup \{(i,l),(l,i) \mid l \in D\}$
8: **end for**
9: **return** $S$

pairs is initialised. The set $C$ is constructed of indices for those variables whose $T_1$-symmetry relation with $x_i$ is as yet undetermined. The set of $T_1$-symmetric variables $D$ returned from `RemoveAsymmetry` and its complement $C \setminus D$ are used to extend $S$ and $A$ respectively. The main loop only requires $n-1$ iterations because $C = \emptyset$ when $i = n$. The algorithm that initialises $A$ is justified by lemmas that detail how $T_1$-symmetric variables place structural constraints on ROBDDs [MMW93]. We state these lemmas below for completeness:

**Lemma 3.1** (Theorem 3.1 [MMW93])**.** *If an ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_j)$ and $i < j$, then every ROBDD rooted at a node labelled $x_i$ must contain a node labelled $x_j$.*

**Lemma 3.2** (Theorem 3.1 [MMW93])**.** *If an ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_j)$ and $i < j$, then every path from the root of $f$ to a node labelled $x_j$ must visit a node labelled $x_i$.*

Lemmas 3.1 and 3.2 provide two conditions under which asymmetry can be observed. For any given node labelled $x_i$ we can compute the set of all variables $x_j$ that appear in a ROBDD that is rooted at that node, and any variable not appearing in this set is necessarily $T_1$-asymmetric with $x_i$. Furthermore, for any given node labelled $x_j$, we can compute the set of all variables $x_i$ that appear on *all* paths from the root of the ROBDD to the node, and any variable not appearing in this set is $T_1$-asymmetric with $x_j$. These asymmetry conditions can be checked together in just two depth-first traversals of the ROBDD, each traversal taking $O(n|G|)$ time since each node is visited singly and at most $n$ variables need be considered.

The symmetry relations between the variables are computed in a series of passes. The validity of this decomposition is justified by the proposition:

**Proposition 3.1.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_j)$ and $i < j$ iff*

- *every ROBDD rooted at a node labelled $x_i$ is $T_1$-symmetric in $(x_i, x_j)$ and,*

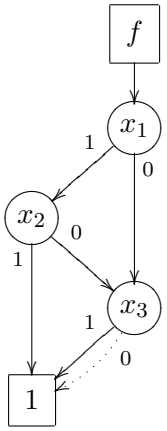


Figure 6: ROBDD of $f = (x_1 \wedge x_2) \vee x_3$

- *every path from the root of $f$ to a node labelled $x_j$ passes through a node labelled $x_i$.*

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Since $f$ is $T_1$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ hence $g|_{x_i \leftarrow 1, x_j \leftarrow 0} = g|_{x_i \leftarrow 0, x_j \leftarrow 1}$.
- Suppose for the sake of a contradiction that there exists a path from the root to a node labelled $x_j$ that does not pass through a node labelled $x_i$. Thus, let $g = f(\mathbf{b}_1, 0, \mathbf{b}_2, x_j, \ldots, x_n) = f(\mathbf{b}_1, 1, \mathbf{b}_2, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$. Thus $g|_{x_j \leftarrow 0}(\mathbf{b}_3) = g|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for all $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = g|_{x_j \leftarrow 1}$ which is a contradiction since $g$ is reduced.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$. Hence $g$ is labelled $x_i$. Thus there exists some $\mathbf{b}_2$ and $\mathbf{b}_3$ such that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 1, \mathbf{b}_3) = 0$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$. Hence $g$ is not labelled $x_i$. Furthermore, let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ as required.

The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ case follows analogously. □

One may wonder if the second condition in the proposition is actually necessary. Figure 6 illustrates that the second condition cannot be relaxed. Observe that the variable

pair $(x_2, x_3)$ is $T_1$-symmetric in the ROBDD rooted at $x_2$, moreover the pair $(x_2, x_3)$ is $T_1$-symmetric for *every* ROBDD rooted at a node labelled $x_2$. However, the pair $(x_2, x_3)$ is $T_1$-asymmetric in the ROBDD $f$, and indeed there exists a path from the root of $f$ to the node $x_3$ that does not visit a node labelled $x_2$. In fact, disabling the preprocessing gives the following asymmetry and symmetry sets $A_i$ and $S_i$ after $i$ iterations of the loop: $A_0 = S_0 = \emptyset$, $A_1 = \{(x_1, x_3), (x_3, x_1)\}, S_1 = \{(x_1, x_2), (x_2, x_1)\}$, $A_2 = A_1, S_2 = S_1 \cup \{(x_2, x_3), (x_3, x_2)\}$. Observe the erroneous pair $(x_2, x_3)$ contained within $S_2$.

The proposition allows exhaustive checking to be decomposed into a series of passes; one pass for each variable $x_i$. Observe that when the loop is entered in Algorithm 1, `FindAsymmetry` has already added all the pairs $(i, j)$ to $A$ such that there exists a path from the root to a node labelled $x_j$ which does not pass through a node labelled $x_i$. An index $j$ for such a pair cannot arise in $C$. Hence it remains to remove those indices $j \in C$ which violate the first condition of the proposition, that is, those $j \in C$ for which $f$ is $T_1$-asymmetric in the pair $(x_i, x_j)$. This is precisely the rôle of `RemoveAsymmetry`$(f, i, C)$ in Algorithm 2 where the parameter $i$ delineates the variable under consideration in the pass. The algorithm uses the function $index(f)$ which merely returns the index of the root of an ROBDD $f$, that is, $i$ if the root of $f$ is labelled $x_i$.

**Algorithm 2** RemoveAsymmetry$(f, i, C)$

**Require:** $f \in ROBDD_X, i \in \mathbb{N} \cup \{0\}$ and $C \subset X$
1: **if** $C = \emptyset \vee f = true \vee f = false$ **then**
2: **return** $C$
3: **end if**
4: $j \leftarrow index(f)$
5: **if** $j > i$ **then**
6: **return** $C$
7: **else if** $j = i$ **then**
8: **return** `RemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$
9: **else**
10: $C \leftarrow$ `RemoveAsymmetry`$(f|_{x_j \leftarrow 0}, i, C)$
11: **return** `RemoveAsymmetry`$(f|_{x_j \leftarrow 1}, i, C)$
12: **end if**

An index $j$ should be removed from $C$ whenever $f|_{x_i \leftarrow 0, x_j \leftarrow 1} \neq f|_{x_i \leftarrow 1, x_j \leftarrow 0}$. This $T_1$-asymmetry check is satisfied if there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ such that,

$$f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, x_{j+1}, \ldots, x_n) \quad \neq \quad f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, x_{j+1}, \ldots, x_n)$$

If $f_0 = f|_{x_j \leftarrow 0}$ and $f_1 = f|_{x_j \leftarrow 1}$ this amounts to showing one of the following,

$$\begin{aligned} f_0(\mathbf{b}_1, 0, \mathbf{b}_2, 1, x_{j+1}, \ldots, x_n) &\neq f_0(\mathbf{b}_1, 1, \mathbf{b}_2, 0, x_{j+1}, \ldots, x_n) \\ f_1(\mathbf{b}_1, 0, \mathbf{b}_2, 1, x_{j+1}, \ldots, x_n) &\neq f_1(\mathbf{b}_1, 1, \mathbf{b}_2, 0, x_{j+1}, \ldots, x_n) \end{aligned}$$

for some (smaller) $\mathbf{b}_1 \in \mathbb{B}^{i-2}$. This recursive reduction explains the recursive nature of `RemoveAsymmetry`. The test $j > i$ implements a form of early termination since if $j > i$ there is no opportunity for removing any index from $C$. The leaf nodes *true* and *false* also trigger early termination.

**Algorithm 3** RemoveAsymmetryVar$(g_0, g_1, C)$

**Require:** $g_0 \in ROBDD_X, g_1 \in ROBDD_X$ and $C \subset X$
1: **if** $g_0 = true \vee g_0 = false$ **then**
2: $\quad j \leftarrow \infty$
3: **else**
4: $\quad j \leftarrow index(g_0)$
5: **end if**
6: **if** $g_1 = true \vee g_1 = false$ **then**
7: $\quad k \leftarrow \infty$
8: **else**
9: $\quad k \leftarrow index(g_1)$
10: **end if**
11: **if** $C = \emptyset \vee j = k = \infty$ **then**
12: $\quad$ **return** $C$
13: **else if** $j = k$ **then**
14: $\quad (l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (j, g_0|_{x_j \leftarrow 0}, g_0|_{x_j \leftarrow 1}, g_1|_{x_k \leftarrow 0}, g_1|_{x_k \leftarrow 1})$
15: **else if** $j < k$ **then**
16: $\quad (l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (j, g_0|_{x_j \leftarrow 0}, g_0|_{x_j \leftarrow 1}, g_1, g_1)$
17: **else**
18: $\quad (l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (k, g_0, g_0, g_1|_{x_k \leftarrow 0}, g_1|_{x_k \leftarrow 1})$
19: **end if**
20: **if** $g_{01} \neq g_{10}$ **then**
21: $\quad C \leftarrow C \setminus \{l\}$
22: **end if**
23: $C \leftarrow$ `RemoveAsymmetryVar`$(g_{00}, g_{10}, C)$
24: **return** `RemoveAsymmetryVar`$(g_{01}, g_{11}, C)$

At the heart of `RemoveAsymmetry` is a call to `RemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$ which is applied to an ROBDD whose root is labelled with the variable $x_i$. When a call to `RemoveAsymmetryVar` is initially encountered, its first and second parameters are $g_0 = g|_{x_i \leftarrow 0}$ and $g_1 = g|_{x_i \leftarrow 1}$. At this point, it remains to search for some $\mathbf{b} \in \mathbb{B}^{j-i-1}$ such that $g_0(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) \neq g_1(\mathbf{b}, 0, x_{j+1}, \ldots, x_n)$. This is in turn realised by showing

either one of the following,

$$\begin{aligned} g_{00}(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) &\neq g_{10}(\mathbf{b}, 0, x_{j+1}, \ldots, x_n) \\ g_{01}(\mathbf{b}, 1, x_{j+1}, \ldots, x_n) &\neq g_{11}(\mathbf{b}, 0, x_{j+1}, \ldots, x_n) \end{aligned}$$

for some (smaller) $\mathbf{b} \in \mathbb{B}^{j-i-2}$ where $g_{00} = g_0|_{x_{i+1} \leftarrow 0}, g_{10} = g_1|_{x_{i+1} \leftarrow 0}, g_{01} = g_0|_{x_{i+1} \leftarrow 1}$ and $g_{11} = g_1|_{x_{i+1} \leftarrow 1}$. A recursive formulation of `RemoveAsymmetryVar` can be obtained from this recursive reduction. When both $g_0$ and $g_1$ are leaf nodes, no further reduction can be applied and `RemoveAsymmetryVar` terminates. The three cases in Algorithm 3 are required to accommodate the reduction inherent in ROBDDs. The $j = k$ condition selects the case when $g_0$ and $g_1$ are labelled with the same variable $x_j$. In this case we compute $g_0|_{x_j \leftarrow 1}$ and $g_1|_{x_j \leftarrow 0}$ and check that $g_0|_{x_j \leftarrow 1} \neq g_1|_{x_j \leftarrow 0}$. If the check is satisfied $j$ is removed from $C$. When $j < k$ the co-factor $g_1|_{x_j \leftarrow 1} = g_1$ hence the asymmetry check $g_0|_{x_j \leftarrow 1} \neq g_1|_{x_j \leftarrow 0}$ reduces to $g_0|_{x_j \leftarrow 1} \neq g_1$. If this check is satisfied $j$ is removed from $C$. The $k < j$ case is analogous except that $k$ is removed.

Caching can be applied to ensure that the function `RemoveAsymmetryVar` is not called twice on the same pair of ROBDDs $g_0$ and $g_1$. Moreover, the complexity of a call to `RemoveAsymmetryVar` is $O(|G|^2)$. This follows since $C$ can be represented as an array of $n$ Booleans. Then computing $C \setminus \{l\}$ is in $O(1)$, as is the test $C = \emptyset$ when $C$ is augmented with a counter to record $|C|$. Overall, `RemoveAsymmetryVar` can only be invoked a total of $|G|$ times from within Algorithm 1, thus `RemoveAsymmetryVar` contributes $O(|G|^3)$ to the overall running time. The $n - 1$ calls to `RemoveAsymmetry` cumulatively cost $O(n|G|)$. Returning to the main loop of Algorithm 1, observe that the sets $A$ and $S$ can be augmented in $O(n)$ time when $D$ is also represented as an array of $n$ Booleans and $A$ and $S$ are represented as $n \times n$ adjacency matrices. Algorithm 1 is therefore in $O(n^2 + n|G| + |G|^3)$. Interestingly, although this improves on the algorithm of Mishchenko when set operations are considered, it does not improve on the naïve co-factor computation method [MMW93, SMMD99] which resides in $O(n^2|G| \log(|G|))$.

## 3.7 Optimised Anytime Symmetry Detection

In this section we propose a series of optimisations for Algorithm 1. The resulting refined algorithm retains the incremental nature of the original algorithm, and shows how incrementality can be exploited by several optimisations. These optimisations seek to reduce the size of the set $C$, and hence the running time of the call `RemoveAsymmetry(`$f, i, C$`)`, by enriching the sets $A$ and $S$ on-the-fly before, and between, iterations of the main loop. The symmetry sieve algorithms proposed by [MMW93, MM93, SMMD99] suggest a way to refine the sets $A$ and $S$ before the loop is entered. Furthermore, it is possible to take advantage of the transitivity of the $T_1$-symmetry relation to add further pairs to $A$ and $S$ between iterations. The optimised algorithm listed in Algorithm 4 takes an

ROBDD $f$ and returns the set $S$ of $T_1$-symmetric variable pairs.

**Algorithm 4** OptimizedSymmetricPairs($f$)

**Require:** $f \in ROBDD_X$
1: $A' \leftarrow$ `FindAsymmetry`$(f)$
2: $M \leftarrow$ `SatisfyCounts`$(f)$
3: **for** $i = 1$ **to** $n$ **do**
4: **for** $j = i + 1$ **to** $n$ **do**
5: **if** $M(i) \neq M(j)$ **then**
6: $A' \leftarrow A' \cup \{(i, j), (j, i)\}$
7: **end if**
8: **end for**
9: **end for**
10: $(A, S) \leftarrow$ `FindAdjSymmetry`$(f)$
11: $(A, S) \leftarrow (A \cup A', S \setminus A')$
12: **for** $i = 1$ **to** $n - 2$ **do**
13: $(A, S) \leftarrow$ `SymmetryClosure`$(A, S)$
14: $C \leftarrow \{ j \mid (i, j) \notin (A \cup S) \wedge i + 1 < j\}$
15: $D \leftarrow$ `RemoveAsymmetry`$(f, i, C)$
16: $A \leftarrow A \cup \{(i, l), (l, i) \mid l \in C \setminus D\}$
17: $S \leftarrow S \cup \{(i, l), (l, i) \mid l \in D\}$
18: **end for**
19: **return** $S$

`SatisfyCounts`$(f)$ returns a mapping $M$ from variable indices to a natural number that can be used to distinguish pairs of $T_1$-asymmetric variables, that is, if $M(i) \neq M(j)$ then $(x_i, x_j)$ are $T_1$-asymmetric. `FindAdjSymmetry`$(f)$ returns two sets of index pairs $A$ and $S$ where $\{(i, j) \mid \neg {T_1}^{x_i,x_j}(f) \wedge j = i + 1\} \subseteq A \subseteq \{(i, j) \mid \neg {T_1}^{x_i,x_j}(f)\}$ and $S = \{(i, j) \mid {T_1}^{x_i,x_j}(f) \wedge j = i + 1\}$. Since the procedure `FindAdjSymmetry` finds all adjacent $T_1$-symmetric and $T_1$-asymmetric pairs, the number of loop iterations can be relaxed from $n-1$ to $n-2$. `SymmetryClosure`$(A_1, S_1)$ takes as input two sets $A_1$ and $S_1$ of variable pairs known to be $T_1$-asymmetric and $T_1$-symmetric respectively. Then, by reasoning about transitivity, a pair of sets $(A_2, S_2)$ is computed which are $T_1$-symmetric and $T_1$-asymmetric such that $A_2 \supseteq A_1$ and $S_2 \supseteq S_1$. The procedures `SatisfyCounts`, `FindAdjSymmetry` and `SymmetryClosure` are detailed in Sections 3.7.1, 3.7.2 and 3.7.3 respectively. Section 3.7.4 presents some heuristics which endeavour to increase the proportion of $T_1$-symmetric variable pairs that are discovered early on in the execution of the main loop of Algorithm 4.

### 3.7.1 Satisfy Counts

A consequence of $T_1$-symmetry, which can also be used to detect $T_1$-asymmetry [MM93], relates the satisfy count of one positive co-factor of a variable to the satisfy count of another:

**Lemma 3.3.** *If a Boolean function $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_j)$, then $\|f|_{x_i \leftarrow 1}\| = \|f|_{x_j \leftarrow 1}\|$.*

Computing the satisfy counts of all co-factors can be realised using a single depth-first traversal of the ROBDD in $O(n|G|)$ time [MM93]. Finding the resultant asymmetries additionally requires $n^2$ comparisons in Algorithm 4, and thus the overall complexity of this sieve is $O(n^2 + n|G|)$.

### 3.7.2 Adjacent Symmetries

The following result follows immediately from Proposition 3.1 and details a special case of symmetry which relates to variables that are adjacent in the ROBDD ordering:

**Corollary 3.1.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_{i+1})$ iff,*

- *every ROBDD rooted at a node labelled $x_i$ is $T_1$-symmetric in $(x_i, x_{i+1})$ and,*
- *every path from the root to a node labelled $x_{i+1}$ passes through a node labelled $x_i$.*

Although the result follows from Proposition 3.1, it was in fact inspired by a tactic of Möller *et al.* [MMW93, SMMD99]. They state that an ROBDD $f$ is $T_1$-symmetric in a pair $(x_i, x_{i+1})$ iff $\|f|_{x_i \leftarrow 1}\| = \|f|_{x_{i+1} \leftarrow 1}\|$ and each ROBDD within $f$ that is rooted at a node labelled $x_i$ is also $T_1$-symmetric in the pair. The force of this result is that the equivalence $f|_{x_i \leftarrow 0, x_{i+1} \leftarrow 1} = f|_{x_i \leftarrow 1, x_{i+1} \leftarrow 0}$ can be checked in $O(|G|)$ time for all adjacent variable pairs [MMW93]. Since no argument is given for the correctness of this procedure, we replace this tactic with another that is justified by Corollary 3.1. In fact Proposition 3.1 leads to an additional result that can detect $T_1$-asymmetric variable pairs that are not necessarily adjacent in the variable ordering:

**Corollary 3.2.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-asymmetric in the pair $(x_i, x_k)$ if there exists a node $g$ in $f$ labelled $x_i$ with successor nodes labelled $x_k$ and $x_l$ where $i + 1 < k \leq l$ and $g|_{x_i \leftarrow 0, x_k \leftarrow 1} \neq g|_{x_i \leftarrow 1, x_k \leftarrow 0}$.*

These non-consecutive $T_1$-asymmetric pairs can be detected in $O(|G|)$ time. Of course, the first $O(|G|)$ tactic for enriching $A$ and $S$ can only be deployed in conjunction with `FindAsymmetry`; the second tactic is independent of `FindAsymmetry`.

### 3.7.3 Symmetry Closure

The following lemma can be obtained by recalling that a function $f$ remains unchanged under the switching of any pair of $T_1$-symmetric variables:

**Lemma 3.4.** *If a Boolean function $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pairs $(x_i, x_j)$ and $(x_j, x_k)$ then $f$ is also $T_1$-symmetric in the pair $(x_i, x_k)$.*

This transitivity result provides a way of enriching the set $S$, that is, if $(x_i, x_j), (x_j, x_k) \in S$ then it follows that $(x_i, x_k)$ is also a $T_1$-symmetric pair, hence $S$ can be enriched with $(x_i, x_k)$. Further, if $(x_i, x_j) \in S, (x_i, x_k) \in A$ then it follows that the pair $(x_j, x_k)$ is $T_1$-asymmetric, that is, $A$ can be enriched with $(x_j, x_k)$. This follows since if $(x_j, x_k)$ is $T_1$-symmetric then by the lemma it follows that $(x_i, x_k)$ is $T_1$-symmetric, which is a contradiction. Adding those variable pairs to $A$ and $S$ which can be inferred through transitivity is not dissimilar to computing the transitive closure of a binary relation. This motivates adapting the Floyd-Warshall [Flo62, War62] all-pairs-shortest-path algorithm to this task which, in turn, leads to Algorithm 5. The complexity of Algorithm 5 is in $O(n^3)$ when $A$ and $S$ are represented as $n \times n$ adjacency matrices since membership check and single element insertion can be performed in $O(1)$ time for an adjacency matrix representation. Each iteration of the main loop of Algorithm 4 incurs an additional call to `SymmetryClosure` which itself resides in $O(n^3)$ and hence pushes the overall complexity into $O(n^4 + n|G| + |G|^3)$. Recall that `SatisfyCounts` and `FindAdjSymmetry` are in $O(n|G|)$ and $O(|G|)$ respectively which have no impact on the overall asymptotic complexity. However, although the Floyd-Warshall algorithm is attractive because of its simplicity, the overall complexity can be reduced to $O(n^3 + n|G| + |G|^3)$, or even lower, by substituting Floyd-Warshall with an incremental (online) transitive closure algorithm [IK83]. It should be noted that the algorithm resides in $O(n^3 + n|G| + |G|^3)$ irrespective of the size and structure of the set $S = \{(i, j) \mid {T_1}^{x_i, x_j}(f)\}$. However, the larger the set $S$, the greater the opportunity for applying transitivity which, in turn, reduces the time spent in `RemoveAsymmetry` and the main loop.

**Algorithm 5** SymmetryClosure$(A, S)$

**Require:** $S, A \in X^2$
1: **for** $i = 1$ **to** $n$ **do**
2:  **for** $j = i + 1$ **to** $n$ **do**
3:  **for** $k = 1$ **to** $n$ **do**
4:  **if** $(k, i) \in S \wedge (k, j) \in S$ **then**
5:  $S \leftarrow S \cup \{(j, i), (i, j)\}$
6:  **else if** $(k, i) \in A \wedge (k, j) \in S$ **then**
7:  $A \leftarrow A \cup \{(j, i), (i, j)\}$
8:  **else if** $(k, i) \in S \wedge (k, j) \in A$ **then**
9:  $A \leftarrow A \cup \{(j, i), (i, j)\}$
10:  **end if**
11:  **end for**
12:  **end for**
13: **end for**
14: **return** $(A, S)$

### 3.7.4 Variable Choice Heuristics

The astute reader may have noticed that the correctness of Algorithm 4 is not compromised by the order in which variables are considered in the main loop. One may wonder therefore if considering variables in a different order can speed up the algorithm. One natural approach is to choose a variable $x_i$ that maximises $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$. The rationale behind this greedy heuristic is to ensure that the call to `RemoveAsymmetry` resolves the maximal number of variable pairs whose $T_1$-symmetry relation is unknown. The dual of this heuristic is to choose a variable $x_i$ for which $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$ is minimal. Motivation for this heuristic comes from literature [MMM02] on computing signatures for Boolean functions so as to determine input correspondence. This is the problem of determining whether the variables of one ROBDD can be reordered so that the resulting ROBDD is equivalent to another. It has been observed that if the currently known asymmetry sieves [MM93,MMW93] leave only a handful of pairs for which a symmetry is unknown, then these variables are likely to be involved in some symmetry relationship [MMM02]. Therefore, focussing `RemoveAsymmetry` on the variable with the least unknowns is likely to discover $T_1$-symmetries. We call these two heuristics max and min respectively. It should be pointed out that for both these heuristics, a variable can be chosen in $O(n)$ time by maintaining a counter for each variable $x_i$ that records the number of unknowns, that is, $|\{(x_i, x_j) \notin (A \cup S) \wedge i < j\}|$. The counter for $x_i$ is decremented each time a pair $(x_i, x_j)$ is added to $A$ or $S$. The cumulative overhead of running the heuristic over the loop body is in $O(n^2)$ which is absorbed into the asymptotic running time of the algorithm.

## 3.8 Experimental Results

To assess the efficiency of the anytime approach, the algorithm and all its refinements, were implemented using the Colorado University Decision Diagram (CUDD) package [Som05]. The rationale for this choice of library was that the Extra DD library [Mis08], which implements Mishchenko's algorithm, also uses CUDD. All Experiments were performed on an UltraSPARC IIIi 900MHz based system, equipped with 16GB RAM, running the Solaris 9 Operating System, using `getrusage` to gauge CPU usage in seconds. All programs — the CUDD package, the Extra library, and our algorithm — were compiled with the GNU C Compiler version 3.3.0 with `-O3` enabled. The algorithms were run against a range of MCNC and ISCAS benchmark circuits of varying size [BEN08], as well as several other benchmarks derived from the SAT literature. All timings are given in seconds and averaged over four runs.

Tables 2 and 3 present the results of these tests. Table 2 presents results where ROBDD construction was performed under a static variable order, i.e. without dynamic variable ordering. By way of contrast, Table 3 presents results where ROBDD

construction was performed in conjunction with the dynamic variable ordering algorithm of Rudell [Rud93]. Hence Table 3 illustrates the impact of dynamic variable ordering in the reading and construction of the ROBDDs and the time required to compute classical symmetries. In both tables, the first four columns give, respectively, the circuit name, number of input variables, number of defined functions (outputs) and the total number of internal ROBDD nodes required to represent all outputs. Column five indicates the total number of all $T_1$-symmetric pairs found over all the outputs. Column six gives the time in seconds to read in the benchmark circuit and construct the ROBDD. The remaining seven columns give the run times required to compute all $T_1$-symmetric and $T_1$-asymmetric pairs. The first of these is the naïve method for computing all co-factor pairs. The second applies the sieves of §3.7.1 and §3.7.2 to reduce the number of co-factors that need to be computed. The third and fourth columns are Mishchenko's implementation of his own algorithm [Mis08] without and with garbage collection enabled. The fifth column is the unoptimised anytime algorithm presented in Section 3.6. The remaining three columns relate to the refinements presented in Section 3.7, that is, with the optimisations of Sections 3.7.1, 3.7.2 and 3.7.3 cumulatively enabled. The rationale for implementing the naïve method was to verify the implementation of our algorithm and Mishchenko's; the performance numbers are included to quantify the value of Mishchenko's algorithm. It is interesting to observe that the differences between running with garbage collection turned on and off are considerable and unpredictable. This slow down is not an anomaly due to garbage generated in an earlier experiment since each experiment is run as a separate process. The garbage stems from the way Mishchenko's implementation makes extensive use of ZBDDs [Min93] to represent sets. Enabling garbage collection has no perceivable impact on our algorithm.

The columns labelled Sat, Adj and Close suggest that all the optimisations to the basic anytime algorithm are worthwhile, though not essential. Interestingly, computing transitive closure is not prohibitively expensive even when implemented using the sub-optimal Floyd-Warshall algorithm. This is because this algorithm can be implemented efficiently and straightforwardly with three nested loops. The simplicity of this optimisation suggests that it should be applied in conjunction with the naïve method [MMW93].

Tables 2 and 3 can only be meaningfully interpreted in conjunction with asymptotic complexity results. Complexity results, such as the assertion that the basic anytime algorithm resides in $O(|G|^3)$ assuming $n \leq |G|$, are ultimately statements about scalability; such results predict how the running time of an algorithm will grow with the size of the input ROBDD. These statements have particular weight when combined with the experimental results of Tables 2 and 3 that gauge the asymptotic constants. For instance, if the basic anytime algorithm terminates within an acceptable time for very large ROBDDs then (no matter whether the ROBDD has been created with or without

sifting, and irrespective of the number of symmetries inferred), the algorithm will terminate within an acceptable time for smaller ROBDDs. This is because the total number of atomic operations is $O(|G|)$. Interestingly, the algorithm of Mishchenko is $O(|G|^3)$ in the number of set operations, where each set operation will have variable complexity depending, for instance, on the number of represented symmetry pairs. Moreover, when sets are realised as ZBDDs, the cost of each set operation will also vary due to memoisation (caching) effects and the overheads induced by memory management. This variability is evident in the columns Mish-GC and Mish+GC. This key difference in the asymptotic complexity explains why, although the running time of the anytime algorithms are consistently below 200 secs, and certainly never exceeds 2 hours, that these algorithms are not uniformly faster than the algorithm of Mishchenko because of the variability of its ZBDD operations.

One may wonder how the performance of the classical anytime algorithm is affected by the underlying architecture. Table 4 thus summarises the results of some timing experiments performed with an Intel Core2 Duo 2.33GHZ PC (using just one core), equipped with 2GB of RAM, running MacOSX. The Intel is faster than the UltraSPARC, but the memory limit of 2GB prevents some circuits (including all those for the larger SAT benchmarks) from being constructed. As before, the running times of the ZBDDs based algorithms are more variable than those of the anytime algorithms. It should be noted the relative timings of the algorithms may change even between Intel machines, due to different memory speeds and caching behaviour.

Figure 7 summarises the outcome of some experiments that investigate the relationship between the variable choice heuristics and the proportion of symmetries found early in the execution of the algorithm. The graphs display the number of symmetries found against various timeouts for the min and max heuristics using the original algorithm as a control. Apart from the circuits *cnt08*, *homer08*, *rope_0006*, *urquhart4_25* (graphs 6, 16, 18 and 20) the min heuristic increases the proportion of symmetries found early in the execution of the algorithm. In the case of *dp02s02* (graph 9), *gripper12* (graph 14) and *homer06* (graph 15) the difference between min and both the control and max is stark. This suggests that the min heuristic should always be applied since it never gives a significant slowdown when the algorithm is run to completion and is beneficial in the case of early termination. In approximately half of the circuits the number of symmetries grows consistently with time. However, for the remainder, growth is either more sporadic or biased towards the latter passes of the symmetry detection algorithm. For these circuits, only a fraction of symmetry pairs could be recovered if these algorithms were terminated prematurely. This is why it is important that anytime generality should not be achieved at the expense of efficiency.

## 3.9 Conclusions

In this chapter we have presented a novel anytime classical symmetry detection algorithm, that is capable of detecting all $T_1$-symmetric variable pairs. The startling speed-ups over Mishchenko's algorithm stem from our use of a single static adjacency matrix rather than sets of pairs that are repeatedly generated. It is important to appreciate that there is no obvious way to re-engineer Mishchenko's algorithm to use a static adjacency matrix. This is because Mishchenko's algorithm is a bottom-up, divide and conquer algorithm that derives the solution to a problem by obtaining, and combining, the solutions to several sub-problems. Mishchenko [Mis03, p 1590] points out that caching of the answers to these sub-problems is required to reduce the computational complexity from exponential to polynomial yet this requires multiple data structures to be maintained. By contrast, the anytime approach merely has to mark nodes as visited in any of the ROBDD traversals. Moreover, the only set operations that the anytime algorithm require are atomic $O(1)$ insertions and deletions, which finesses the otherwise $O(n^2)$ overhead of set intersection and union. This partly explains the speed of the anytime approach.

| Circuit | # In | # Out | Σ\|G\| | \|S\| | Read | Naïve | Muller | Mish-GC | Mish+GC | Any | Sat | Adj | Close |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pair | 173 | 137 | 118066 | 1910 | 0.20 | 132.46 | 4.45 | 6.62 | 35.50 | 2.37 | 2.18 | 2.16 | 2.08 |
| C3540 | 50 | 22 | 4618194 | 81 | 21.80 | >7200 | 122.09 | 132.72 | 5488.75 | 71.64 | 68.23 | 66.08 | 65.04 |
| C880 | 60 | 26 | 600998 | 262 | 8.29 | 704.54 | 10.23 | 13.90 | 2242.11 | 7.75 | 6.84 | 5.63 | 5.20 |
| s4863 | 153 | 104 | 126988 | 547 | 2.63 | 20.60 | 1.45 | 5.30 | 5.71 | 1.41 | 1.08 | 1.01 | 0.82 |
| s9234.1 | 247 | 250 | 4434504 | 3454 | 20.14 | >7200 | 1415.88 | 1407.20 | >7200 | 183.84 | 158.36 | 145.94 | 141.26 |
| s38584.1 | 1464 | 1730 | 150554 | 15629 | 3.70 | 337.59 | 23.01 | 16.70 | 132.16 | 3.12 | 3.04 | 3.01 | 2.80 |
| simp10 | 105 | 1 | 722074 | 19 | 58.45 | >7200 | 186.05 | 661.70 | >7200 | 65.28 | 47.53 | 43.90 | 40.88 |
| simp12 | 117 | 1 | 758330 | 23 | 76.23 | >7200 | 139.45 | >7200 | >7200 | 105.67 | 61.94 | 59.87 | 57.59 |
| simp14 | 120 | 1 | 562326 | 36 | 70.38 | >7200 | >7200 | 1114.29 | >7200 | 75.75 | 38.48 | 36.17 | 30.63 |
| hom06 | 104 | 1 | 1176845 | 20 | 65.22 | >7200 | 443.67 | 274.90 | >7200 | 115.66 | 91.70 | 88.31 | 81.50 |
| hom08 | 95 | 1 | 893312 | 16 | 56.48 | >7200 | 466.21 | 135.79 | >7200 | 67.79 | 54.99 | 50.89 | 49.00 |
| hom10 | 130 | 1 | 309221 | 29 | 29.98 | >7200 | 446.56 | 1510.32 | >7200 | 35.85 | 33.39 | 31.61 | 31.21 |
| ca004 | 53 | 1 | 782640 | 2 | 5.40 | >7200 | 15.01 | 147.97 | >7200 | 31.35 | 12.33 | 12.33 | 12.10 |
| ca008 | 96 | 1 | 682617 | 16 | 20.40 | >7200 | 250.16 | 326.92 | >7200 | 53.54 | 44.69 | 43.05 | 42.78 |
| ca016 | 107 | 1 | 861209 | 26 | 60.10 | >7200 | 744.55 | 305.11 | >7200 | 72.68 | 59.96 | 50.90 | 50.80 |
| urquhart2_25 | 48 | 1 | 722657 | 5 | 3.06 | >7200 | 42.95 | 70.50 | >7200 | 26.22 | 20.23 | 20.21 | 17.95 |
| urquhart3_25 | 62 | 1 | 1771025 | 24 | 6.22 | >7200 | 560.18 | >7200 | >7200 | 82.98 | 81.14 | 76.97 | 72.80 |
| urquhart4_25 | 68 | 1 | 1736705 | 27 | 5.96 | >7200 | 974.83 | >7200 | >7200 | 83.44 | 81.84 | 76.48 | 72.02 |
| rope_0002 | 54 | 1 | 634914 | 3 | 3.06 | >7200 | 19.08 | 192.77 | >7200 | 22.48 | 18.53 | 18.47 | 18.50 |
| rope_0004 | 62 | 1 | 1052214 | 10 | 4.73 | >7200 | 253.82 | 487.26 | >7200 | 41.71 | 39.70 | 37.90 | 37.82 |
| rope_0006 | 61 | 1 | 759039 | 13 | 3.14 | >7200 | 225.23 | 657.74 | >7200 | 35.78 | 30.76 | 30.64 | 30.68 |
| ferry8 | 111 | 1 | 290127 | 30 | 78.35 | >7200 | 605.96 | 95.15 | >7200 | 30.10 | 29.56 | 23.21 | 22.99 |
| ferry10 | 116 | 1 | 539419 | 38 | 88.08 | >7200 | 2177.43 | 1866.62 | >7200 | 70.34 | 69.84 | 54.19 | 53.42 |
| ferry12 | 123 | 1 | 277291 | 36 | 47.96 | >7200 | 988.06 | 142.10 | >7200 | 37.63 | 37.50 | 30.98 | 30.95 |
| gripper10 | 125 | 1 | 393485 | 28 | 69.08 | >7200 | 1641.05 | 261.32 | >7200 | 52.97 | 50.53 | 45.38 | 44.74 |
| gripper12 | 129 | 1 | 667877 | 43 | 50.95 | >7200 | 2604.07 | 368.50 | >7200 | 106.32 | 102.87 | 85.43 | 84.90 |
| gripper14 | 118 | 1 | 767735 | 40 | 47.29 | >7200 | >7200 | 415.57 | >7200 | 111.49 | 110.40 | 73.48 | 71.34 |

Table 2: $T_1$-symmetry Experimental Results without Sifting

| **Circuit** | **# In** | **# Out** | **Σ\|G\|** | **\|S\|** | **Read** | **Naïve** | **Muller** | **Mish-GC** | **Mish+GC** | **Any** | **Sat** | **Adj** | **Close** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| pair | 173 | 137 | 8599 | 1910 | 0.60 | 2.71 | 0.50 | 0.18 | 0.62 | 0.48 | 0.36 | 0.32 | 0.28 |
| C3540 | 50 | 22 | 43334 | 81 | 14.00 | 38.37 | 0.99 | 0.94 | 6.84 | 3.45 | 2.89 | 2.35 | 1.99 |
| C880 | 60 | 26 | 8753 | 262 | 0.44 | 5.20 | 0.13 | 0.22 | 1.01 | 0.24 | 0.16 | 0.12 | 0.10 |
| s4863 | 153 | 104 | 75549 | 547 | 87.58 | 14.78 | 0.80 | 0.09 | 1.28 | 0.50 | 0.32 | 0.29 | 0.16 |
| s9234.1 | 247 | 250 | 9376 | 3454 | 2.16 | 6.76 | 0.76 | 0.39 | 1.46 | 0.87 | 0.74 | 0.68 | 0.42 |
| s38584.1 | 1464 | 1730 | 34833 | 15629 | 13.10 | 18.36 | 1.72 | 2.89 | 4.11 | 4.83 | 3.26 | 2.96 | 2.80 |
| simp10 | 105 | 1 | 222431 | 19 | 205.11 | >7200 | 11.22 | 4.72 | 57.17 | 32.53 | 11.65 | 11.13 | 11.12 |
| simp12 | 117 | 1 | 292811 | 23 | 230.61 | >7200 | 22.19 | 12.61 | 61.96 | 55.55 | 22.22 | 21.81 | 21.96 |
| simp14 | 120 | 1 | 86267 | 36 | 111.84 | >7200 | 7.49 | 16.24 | 121.69 | 11.98 | 7.16 | 7.14 | 7.10 |
| hom06 | 104 | 1 | 60357 | 20 | 170.76 | >7200 | 3.81 | 8.98 | 9.65 | 6.56 | 3.70 | 3.77 | 3.85 |
| hom08 | 95 | 1 | 110160 | 16 | 128.91 | >7200 | 4.39 | 4.18 | 134.31 | 17.48 | 4.70 | 4.74 | 4.50 |
| hom10 | 130 | 1 | 142827 | 29 | 283.80 | >7200 | 33.49 | 106.28 | 106.38 | 56.00 | 33.51 | 33.68 | 33.74 |
| ca004 | 53 | 1 | 9119 | 2 | 1.86 | 16.47 | 9.91 | 0.27 | 1.94 | 0.31 | 0.22 | 0.18 | 0.08 |
| ca008 | 96 | 1 | 19945 | 16 | 3.80 | 384.18 | 147.98 | 9.86 | 1235.98 | 1.97 | 1.68 | 1.30 | 1.14 |
| ca016 | 107 | 1 | 90033 | 26 | 33.45 | 6444.37 | 2544.87 | 19.54 | >7200 | 20.19 | 17.01 | 16.36 | 14.10 |
| urquhart2_25 | 48 | 1 | 41098 | 5 | 6.86 | 168.03 | 137.65 | 0.16 | 0.70 | 1.76 | 1.21 | 0.65 | 0.32 |
| urquhart3_25 | 62 | 1 | 43599 | 24 | 3.03 | 1290.93 | 527.77 | 10.66 | >7200 | 4.36 | 3.68 | 3.12 | 2.78 |
| urquhart4_25 | 68 | 1 | 45008 | 27 | 23.21 | 3330.31 | 1070.31 | 4.57 | 3330.31 | 6.94 | 6.37 | 6.31 | 6.23 |
| rope_0002 | 54 | 1 | 1038 | 3 | 0.15 | 2.08 | 0.51 | 0.04 | 0.14 | 0.04 | 0.03 | 0.03 | 0.02 |
| rope_0004 | 62 | 1 | 11874 | 10 | 2.29 | 186.84 | 62.16 | 0.74 | 37.30 | 1.22 | 0.66 | 0.64 | 0.52 |
| rope_0006 | 61 | 1 | 11066 | 13 | 5.01 | 564.39 | 216.53 | 0.40 | 28.17 | 1.28 | 1.03 | 0.99 | 0.98 |
| ferry8 | 111 | 1 | 5998 | 30 | 22.10 | 1890.72 | 791.87 | 4.70 | >7200 | 3.53 | 3.54 | 3.55 | 3.56 |
| ferry10 | 116 | 1 | 3141 | 38 | 6.18 | 140.32 | 64.45 | 0.34 | >7200 | 0.44 | 0.42 | 0.46 | 0.48 |
| ferry12 | 123 | 1 | 3758 | 36 | 21.18 | 785.37 | 331.28 | 0.72 | 386.18 | 1.45 | 1.45 | 1.44 | 1.43 |
| gripper10 | 125 | 1 | 17525 | 28 | 183.67 | >7200 | >7200 | 8.42 | 4348.16 | 39.69 | 37.79 | 37.63 | 39.20 |
| gripper12 | 129 | 1 | 17035 | 43 | 165.65 | >7200 | >7200 | 7.05 | 5365.41 | 35.35 | 34.89 | 34.80 | 36.32 |
| gripper14 | 118 | 1 | 9742 | 40 | 160.23 | >7200 | >7200 | 5.97 | >7200 | 29.53 | 27.10 | 27.23 | 29.68 |

Table 3: $T_1$-symmetry Experimental Results with Sifting

| | with reordering | | | | without reordering | | | |
|---|---|---|---|---|---|---|---|---|
| **Circuit** | **Naïve** | **Möller** | **Mish** | **Close** | **Naïve** | **Möller** | **Mish** | **Close** |
| alu2 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| alu4 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| C1908 | 2.53 | 0.73 | 0.04 | 0.16 | 6.69 | 2.10 | 0.12 | 1.12 |
| C2670 | 35.71 | 5.57 | 0.12 | 0.17 | – | – | – | – |
| C3540 | 13.34 | 0.29 | 0.35 | 0.24 | – | – | – | – |
| C432 | 0.23 | 0.01 | 0.01 | 0.01 | 10.08 | 0.14 | 1.89 | 0.13 |
| C499 | 41.80 | 35.14 | 0.04 | 0.98 | 62.04 | 30.10 | 0.10 | 1.12 |
| C5315 | 3.12 | 0.17 | 0.07 | 0.11 | – | – | – | – |
| C880 | 2.35 | 0.05 | 0.62 | 0.03 | 273.30 | 3.55 | 76.41 | 1.93 |
| dalu | 0.40 | 0.04 | 0.01 | 0.01 | 0.63 | 0.10 | 0.05 | 0.07 |
| des | 0.17 | 0.07 | 0.03 | 0.04 | 0.35 | 0.15 | 0.06 | 0.06 |
| frg2 | 0.10 | 0.02 | 0.01 | 0.01 | 0.24 | 0.03 | 0.04 | 0.02 |
| i10 | 42.85 | 1.43 | 1.36 | 0.56 | 410.74 | 21.13 | 77.45 | 1.96 |
| k2 | 0.37 | 0.01 | 0.03 | 0.01 | 0.36 | 0.02 | 0.04 | 0.01 |
| pair | 1.05 | 0.18 | 0.12 | 0.08 | 46.22 | 1.47 | 5.01 | 0.51 |
| rot | 1.05 | 0.03 | 0.03 | 0.03 | 6.68 | 0.13 | 0.12 | 0.07 |
| s635 | 0.03 | 0.02 | 0.04 | 0.01 | 0.04 | 0.02 | 0.05 | 0.01 |
| s838.1 | 0.07 | 0.02 | 0.04 | 0.01 | 0.08 | 0.02 | 0.05 | 0.01 |
| s1196 | 0.05 | 0.01 | 0.02 | 0.01 | 0.11 | 0.01 | 0.02 | 0.01 |
| s1269 | 0.27 | 0.02 | 0.02 | 0.01 | 0.45 | 0.03 | 0.03 | 0.01 |
| s1423 | 1.26 | 0.10 | 0.07 | 0.08 | 6.86 | 0.58 | 0.15 | 0.13 |
| s3271 | 0.03 | 0.01 | 0.01 | 0.01 | 0.59 | 0.12 | 0.05 | 0.03 |
| s4863 | 5.50 | 0.35 | 0.02 | 0.07 | 9.84 | 0.48 | 0.04 | 0.21 |
| s9234.1 | 1.51 | 0.20 | 0.10 | 0.08 | – | – | – | – |
| too_large | 0.38 | 0.01 | 0.02 | 0.01 | 0.47 | 0.01 | 0.02 | 0.01 |

Table 4: Classical Symmetry Timing Experiments on an Intel

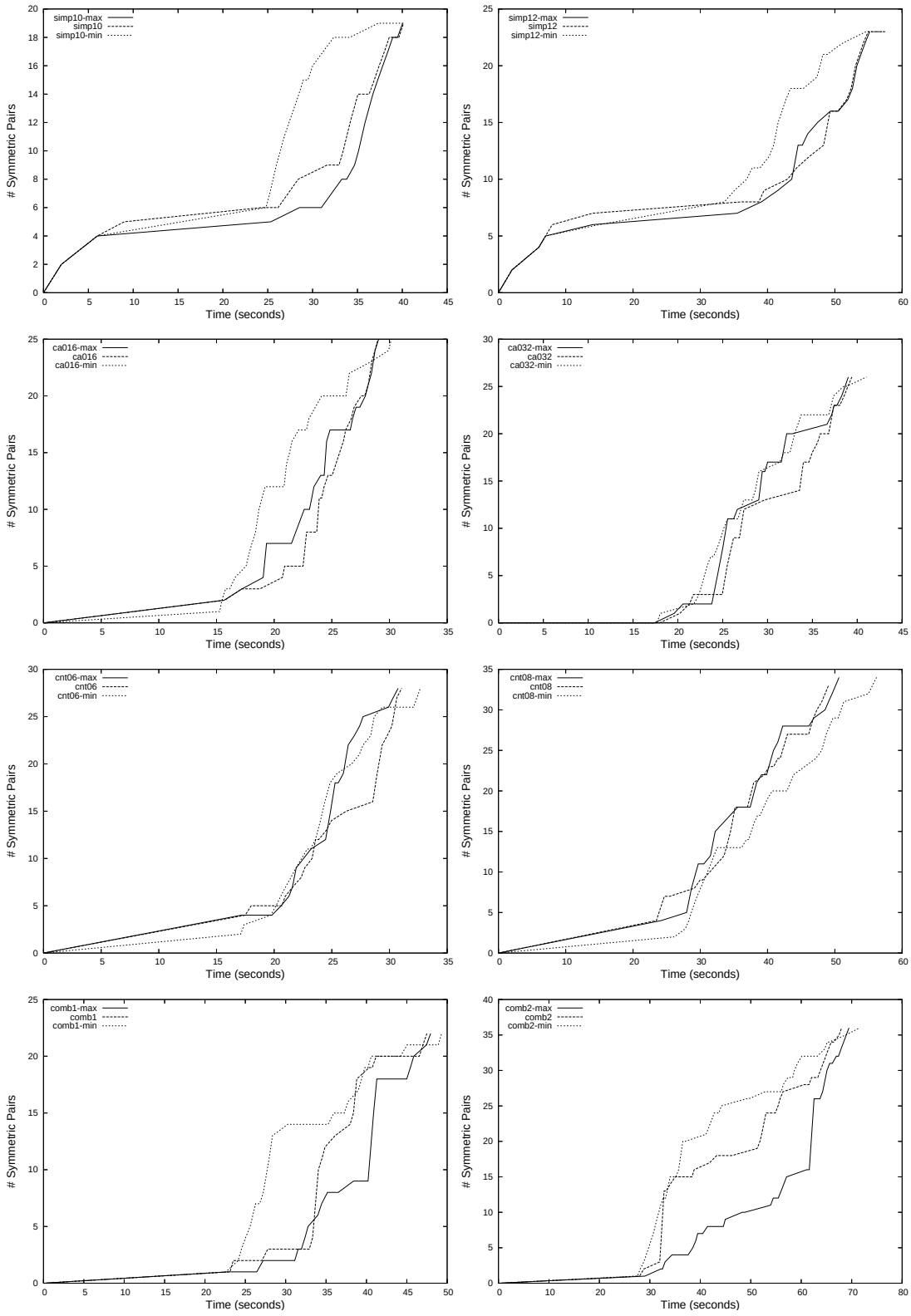


Figure 7: Symmetries against time

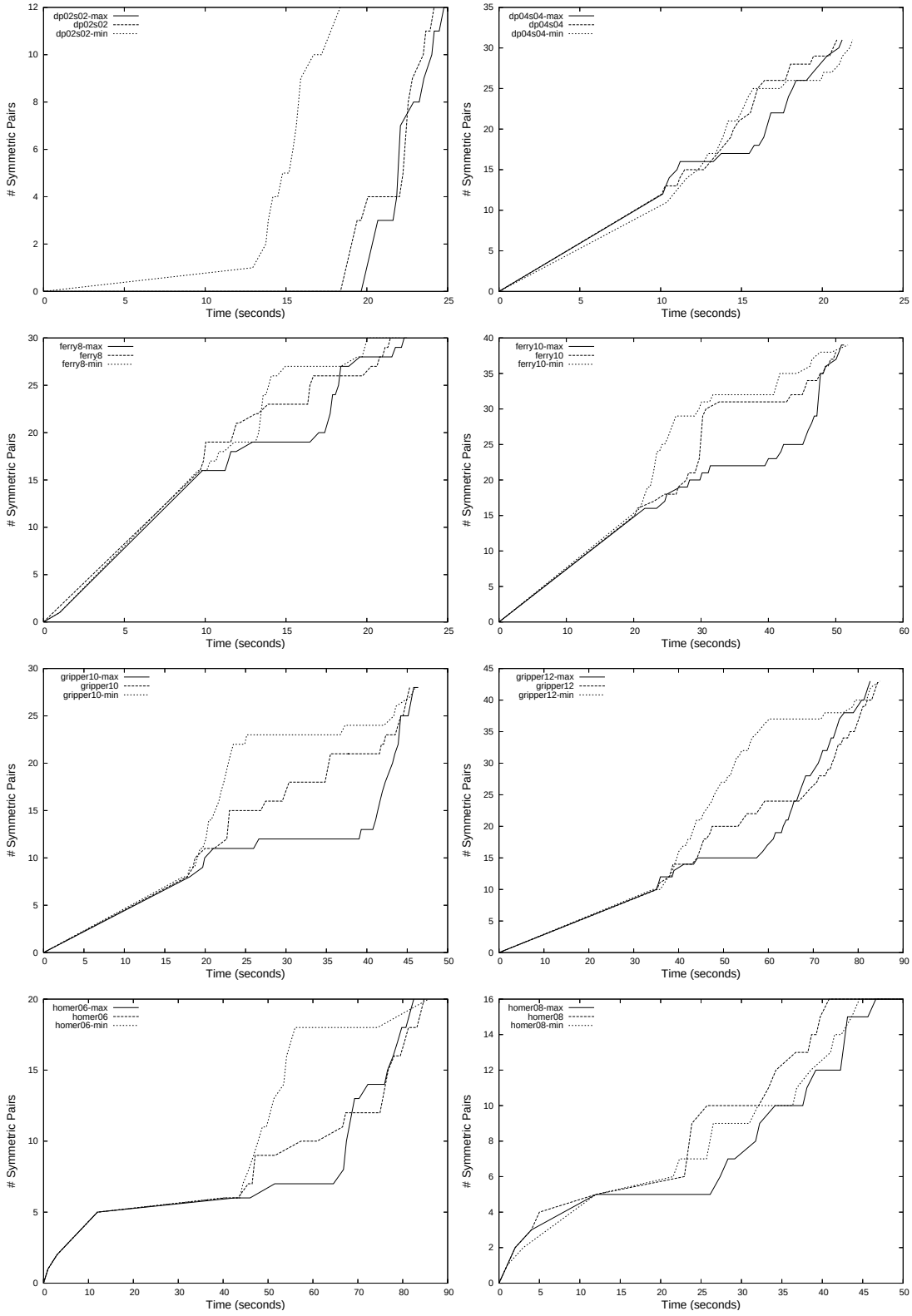


Figure 7: Symmetries against time

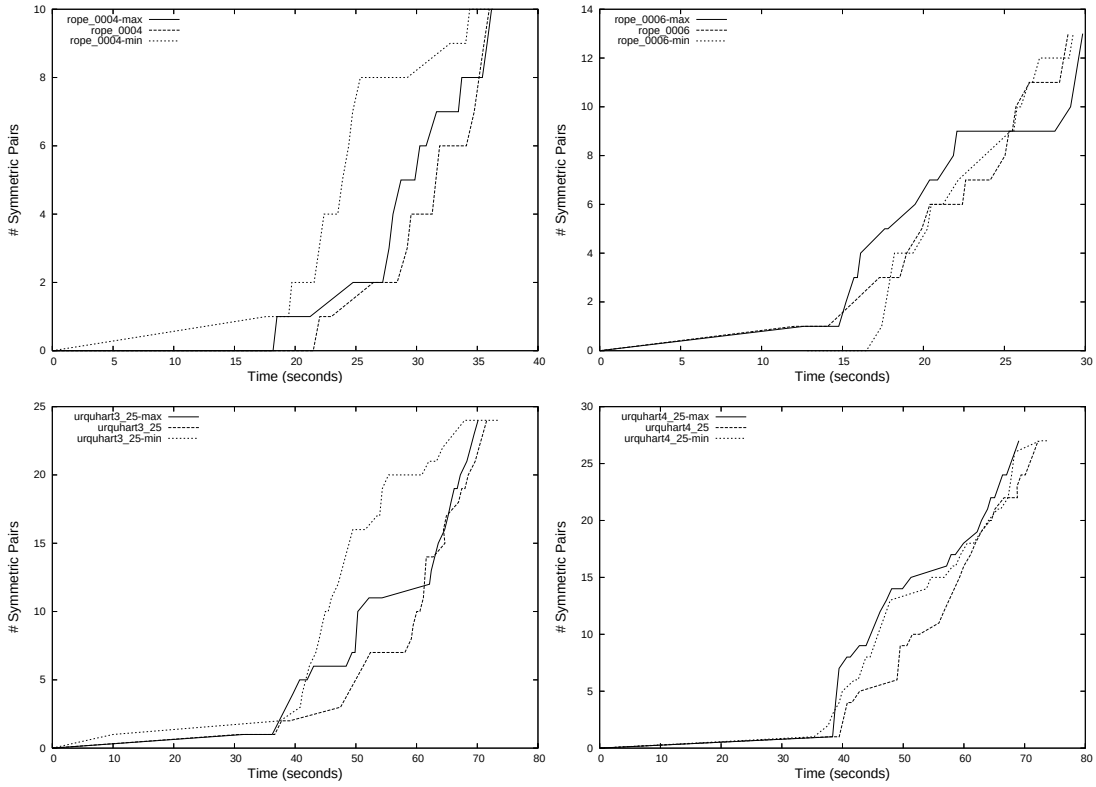


Figure 7: Symmetries against time

# Chapter 4

# Generalised Symmetry Detection in ROBDDs

**Abstract.** In this section we present an anytime algorithm for *generalised* symmetry detection for Boolean functions represented as ROBDDs. The algorithm builds upon the techniques used in the anytime algorithm presented in Chapter 3.

## 4.1 Introduction

In recent years, the notion of classical symmetry has been extended to so-called *generalised* symmetries that cover all possible two co-factor equivalences [KS00b, CJ01, ZCJMB04, ZMBCJ06]. Kravets and Sakallah [KS00b] define so-called higher-order symmetries in which the switching of variable pairs are extended to interchanging groups (or sets) of ordered variables. The concept of generalised symmetry coined by Kravets and Sakallah encompasses all variable permutations under which the function remains unchanged. However, the concept of generalised symmetries considered in this chapter is restricted to those symmetries expressible as a co-factor equivalence on at most two variables: this terminology coincides with that adopted within the majority of the symmetry detection literature [TM97, ZCJMB04, ZMBCJ06]. The generalised symmetry types are listed in Table 5 where $f|_{a,b}$ abbreviates $f|_{x_i \leftarrow a, x_j \leftarrow b}$. These symmetries can be categorised into two types depending on whether or not a negated co-factor occurs in the relationship: $T_1, \ldots, T_6$ coincide with those of Zhang *et al.* [ZCJMB04] whereas $T_7, \ldots, T_{12}$ correspond to the $\neg T_1, \ldots, \neg T_6$ types in the notation of Zhang *et al.*

| Positive Co-factor relations | Negative Co-factor relations |
|---|---|
| ${T_1}^{x_i,x_j}(f) \iff f\|_{1,0} = f\|_{0,1}$ | ${T_7}^{x_i,x_j}(f) \iff f\|_{1,0} = \neg f\|_{0,1}$ |
| ${T_2}^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{1,1}$ | ${T_8}^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{1,1}$ |
| ${T_3}^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{0,1}$ | ${T_9}^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{0,1}$ |
| ${T_4}^{x_i,x_j}(f) \iff f\|_{1,0} = f\|_{1,1}$ | ${T_{10}}^{x_i,x_j}(f) \iff f\|_{1,0} = \neg f\|_{1,1}$ |
| ${T_5}^{x_i,x_j}(f) \iff f\|_{0,0} = f\|_{1,0}$ | ${T_{11}}^{x_i,x_j}(f) \iff f\|_{0,0} = \neg f\|_{1,0}$ |
| ${T_6}^{x_i,x_j}(f) \iff f\|_{0,1} = f\|_{1,1}$ | ${T_{12}}^{x_i,x_j}(f) \iff f\|_{0,1} = \neg f\|_{1,1}$ |

Table 5: Generalised symmetry types

## 4.2 Applications

There exist numerous algorithms designed to detect the classical $T_1$-symmetry type in Boolean functions represented as ROBDDs (see §3.2 and §3.5 for an overview). However, as noted in [ZCJMB04], the often used classical symmetry is actually the least common form of symmetry found in circuits. For example, Table 6 [ZCJMB04] illustrates the distribution of the differing symmetry types for several MCNC benchmark circuits. Overall, the symmetry types $T_1$ and $T_2$ account for only 12.6% of all symmetries found in the benchmarks (indicating only a small number of classical symmetries), while only 0.9% are one of the negated symmetries $T_7$ and $T_8$. However, the symmetry types $T_3, T_4, T_5$ and $T_6$ account for a significant 84.8% of two variable symmetries in the benchmarks, 1.7% are one of the negated symmetries $T_9, T_{10}, T_{11}$ and $T_{12}$-symmetries. The prevalence of generalised symmetries in MCNC benchmarks over classical symmetries motivates the requirement for efficient algorithms to detect them, however, the number of algorithms designed to detect the generalised symmetries is still relatively small. Besides this vacuum that currently exists, the applications of generalised symmetries are much akin to those of classical symmetries (a detailed discussion of these applications can be found in §3.2).

## 4.3 Contributions

In this chapter we present an efficient anytime algorithm for generalised symmetry detection. For clarity, we summarise our contributions as follows:

- We show how to refine the anytime algorithm so as to detect generalised symmetries. An algorithm for simultaneously detecting all $T_1, \ldots, T_{12}$-symmetries is presented which resides in $O(n^3 + n^2|G| + |G|^3)$.
- The algorithm is underpinned by new symmetry relationships which take the form, that if ${T_p}^{x_i,x_j}(f)$ and ${T_q}^{x_j,x_k}(f)$ hold then ${T_r}^{x_i,x_j}(f)$ holds where $T_p, T_q$ and $T_r$ denote one of the 12 generalised symmetry types. Only a few of these transitivity

| **Name** | **Classical** $(T_1, T_2/T_7, T_8)$ | **Single Variable** $(T_3, T_4, T_5, T_6/$ $T_9, T_{10}, T_{11}, T_{12})$ |
|---|---|---|
| 9symml | 36/0 | 0/0 |
| alu2 | 7/1 | 15/6 |
| alu4 | 11/1 | 31/6 |
| apex6 | 408/16 | 2540/29 |
| term1 | 470/2 | 510/0 |
| b1 | 5/3 | 0/4 |
| b9 | 81/8 | 464/9 |
| f51m | 5/8 | 3/67 |
| x1 | 300/6 | 1696/7 |
| c8 | 114/16 | 343/25 |
| i1 | 103/14 | 268/18 |
| z4ml | 22/5 | 0/52 |
| too_large | 20/0 | 482/0 |
| t481 | 8/0 | 16/0 |
| cc | 44/3 | 166/6 |
| k2 | 559/0 | 4191/0 |
| frg1 | 7/3 | 101/4 |
| rot | 540/93 | 5216/99 |
| pm1 | 85/1 | 255/2 |
| Total | 2425/180 | 16297/334 |
| Ratio | 12.6%/0.9% | 84.8%/1.7% |

Table 6: Number of symmetries in MCNC benchmarks [ZCJMB04]

results have been previously reported [TM97] and these results could well find application in other symmetry detection problems [ZMBCJ06].

- We show that generalised symmetry detection does not require the creation of intermediate ROBDDs or even ZBDDs and that anytime generality does not compromise efficiency.

The remainder of this chapter is structured thusly, Section 4.4 presents the necessary preliminaries, §4.5 surveys the related work. Section 4.6 presents an anytime symmetry detection algorithm for generalised symmetries. Sections 4.7 and 4.8 present experimental results and a concluding discussion respectively.

## 4.4 Preliminaries

Each of the 12 predicates $T_i^{x_j,x_k}(f)$ of Table 5 asserts a symmetry property of a Boolean function $f$ where the predicate $T_i^{x_j,x_k}(f)$ is interpreted as stating that the Boolean function $f$ is $T_i$-symmetric in the variable pair $(x_j, x_k)$. Strictly, an ROBDD $g$ is not a Boolean function but rather a representation of one. Therefore to assert symmetry

properties of the function $f$ that underlies a given ROBDD $g$, we define ${T_i}^{x_j,x_k}(g)$ to hold whenever ${T_i}^{x_j,x_k}(f)$ holds. Moreover, we shall say that a ROBDD $g$ is $T_i$-symmetric in the variable pair $(x_j, x_k)$ iff ${T_i}^{x_j,x_k}(g)$ holds, and dually $g$ is $T_i$-asymmetric in the variable pair $(x_j, x_k)$ iff ${T_i}^{x_j,x_k}(g)$ does not hold.

## 4.5 Related Work

The generalisation of symmetries is a recent development and has received much attention [KS00b, CJ01, WKSV03, ZCJMB04]. This move to generalised symmetries has inevitably brought with it the requirement for efficient algorithms to compute them [WKSV03, ZCJMB04]. It is straightforward to extend the naïve approach of symmetry detection to all generalised symmetries in Table 5 with only a worst-case twofold increase in the amount of work required. This is because classical symmetry detection requires calculating the co-factors $f|_{x_i \leftarrow 1, x_j \leftarrow 0}$ and $f|_{x_i \leftarrow 0, x_j \leftarrow 1}$ whereas generalised symmetries over two variables only require the co-factors $f|_{x_i \leftarrow 0, x_j \leftarrow 0}$ and $f|_{x_i \leftarrow 1, x_j \leftarrow 1}$ to be additionally computed. (The amount of work required to compute an equivalence check, such as $f|_{x_i \leftarrow 0, x_j \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 1}$, is negligible and a check that involves negation, such as $f|_{x_i \leftarrow 0, x_j \leftarrow 0} = \neg f|_{x_i \leftarrow 1, x_j \leftarrow 1}$, is also in $O(1)$ for ROBDD packages supporting complement edges [BRB90].) This twofold increase in work is disproportionate to the twelvefold increase in the number of symmetries that can be detected, however, the overhead of repeated co-factoring is still prohibitive. Consequently, symmetry detection methods for generalised symmetries have progressed along the same lines as those for classical symmetries. State-of-the-art in generalised symmetry computation is represented by the algorithm of Zhang *et al.* [ZCJMB04]. The algorithm mirrors the design of Mishchenko [Mis03], but is altered to perform multiple passes for each of the different symmetry types. Hence, the algorithm of Zhang *et al.* has the same worst-case complexity of that of Mishchenko, disregarding constant factors.

## 4.6 Generalised Anytime Symmetry Detection Algorithm

In this section we show how to extend the anytime algorithm presented in the previous chapter to also detect the generalised symmetry types given in Table 5. Algorithm 6 takes as input an ROBDD $f$ and returns the set of triples $S = \{(i, j, k) \mid {T_k}^{x_i,x_j}(f)\}$. The algorithm is composed of three distinct procedures. `FindFastSymmetry`$(f)$ returns a pair $(A, S)$ such that,

$$A = \{(i, j, k) \mid \neg {T_k}^{x_i,x_j}(f) \wedge k \in K\} \quad \text{and}$$
$$S = \{(i, j, k) \mid {T_k}^{x_i,x_j}(f) \wedge k \in K\} \quad \text{where} \quad K = \{3, 4, 9, 10\}$$

`FindSlowAsymmetry`$(f)$ returns a set $A' \subseteq \{(i,j,k) \mid \neg {T_k}^{x_i,x_j}(f) \wedge k \in K'\}$ such that $K' = \{1,\ldots,12\} \setminus K$. In an analogous fashion, `GeneralRemoveAsymmetry`$(f,i,C)$ filters a set of pairs $C$ to return a subset $C' \subseteq C$. If the $T_k$-symmetry relationship between the variables $x_i$ and $x_j$ is presently unknown then $(j,k) \in C$. The returned set $C' \subseteq C$ is precisely those pairs $C' = \{(j,k) \in C \mid {T_k}^{x_i,x_j}(f) \wedge k \in K'\}$.

**Algorithm 6** GeneralizedSymmetricPairs($f$)

**Require:** $f \in ROBDD_X$
1: $(A,S) \leftarrow$ `FindFastSymmetry`$(f)$
2: $A \leftarrow A \cup$ `FindSlowAsymmetry`$(f)$
3: **for** $i = 1$ **to** $n-1$ **do**
4: $\quad C \leftarrow \{\,(j,k) \mid (i,j,k) \notin (A \cup S) \wedge i < j\}$
5: $\quad D \leftarrow$ `GeneralRemoveAsymmetry`$(f,i,C)$
6: $\quad A \leftarrow A \cup \{(i,l,k),\ (l,i,k)\ \mid (l,k)\ \in C \setminus D\}$
7: $\quad S \leftarrow S \cup \{(i,l,k),(l,i,k) \mid (l,k) \in D\}$
8: **end for**
9: **return** $S$

### 4.6.1 Fast Symmetries

Interestingly, some types of generalised symmetry are easier to compute than others. In fact, $T_3$ and $T_4$-symmetries and $T_9$ and $T_{10}$-symmetries can be computed in $O(n|G|)$ and $O(n^2|G|)$ respectively, utilising the following propositions.

**Proposition 4.1.** *An ROBDD $f$ over a set of variables $\{x_1,\ldots,x_n\}$ is $T_3$-symmetric in the pair $(x_i,x_j)$ and $i<j$ iff*

- *if whenever an ROBDD $g$ occurs in $f$ at a node labelled $x_i$ then $g|_{x_i \leftarrow 0}$ does not contain a node labelled $x_j$ and,*
- *every path from the root of $f$ to a node labelled $x_j$ passes through a node labelled $x_i$.*

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Since $f$ is $T_3$-symmetric in the variable pair $(x_i,x_j)$, $f(\mathbf{b}_1,0,\mathbf{b}_2,0,\mathbf{b}_3) = f(\mathbf{b}_1,0,\mathbf{b}_2,1,\mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Let $g = f(\mathbf{b}_1,x_i,\ldots,x_n)$ and $h = g(0,\mathbf{b}_2,x_j,\ldots,x_n)$ thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$. Hence $h|_{x_j \leftarrow 0} = h|_{x_j \leftarrow 1}$, and thus it follows that the ROBDD $h$ cannot be labelled with a variable $x_j$ because $h$ is reduced.
- Suppose for the sake of a contradiction that there exists a path from the root of $f$ to a node $g$ labelled $x_j$ that does not pass through a node labelled $x_i$.

Since the path does not pass through $x_i$, let $g = f(\mathbf{b}_1, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{j-1}$. Since $f|_{x_i \leftarrow 0, x_j \leftarrow 0} = f|_{x_i \leftarrow 0, x_j \leftarrow 1}$ it follows that $g|_{x_j \leftarrow 0}(\mathbf{b}_2) = g|_{x_j \leftarrow 1}(\mathbf{b}_2)$ for all $\mathbf{b}_2 \in \mathbb{B}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = g|_{x_j \leftarrow 1}$, it follows that the ROBDD $g$ cannot be labelled with a variable $x_j$ because $g$ is reduced, hence a contradiction follows.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = 1$ and $h|_{x_j \leftarrow 1}(\mathbf{b}_3) = 0$. Hence $g|_{x_i \leftarrow 0}$ contains a node $h$ labelled $x_j$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = 1$ and $h|_{x_j \leftarrow 1}(\mathbf{b}_3) = 0$. Hence $g$ contains a node labelled $x_j$ and $g$ is not labelled $x_i$ as required.

The $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ case follows analogously. □

**Proposition 4.2.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_4$-symmetric in the pair $(x_i, x_j)$ and $i < j$ iff*

- *if whenever an ROBDD $g$ occurs in $f$ at a node labelled $x_i$ then $g|_{x_i \leftarrow 1}$ does not contain a node labelled $x_j$ and,*
- *every path from the root of $f$ to a node labelled $x_j$ passes through a node labelled $x_i$.*

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Since $f$ is $T_4$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ and $h = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$ thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$. Hence $h|_{x_j \leftarrow 0} = h|_{x_j \leftarrow 1}$, and thus it follows that the ROBDD $h$ cannot be labelled with a variable $x_j$ because $h$ is reduced.
- Suppose for the sake of a contradiction that there exists a path from the root of $f$ to a node $g$ labelled $x_j$ that does not pass through a node labelled $x_i$. Since the path does not pass through $x_i$, let $g = f(\mathbf{b}_1, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{j-1}$. Since $f|_{x_i \leftarrow 1, x_j \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 1}$ it follows that $g|_{x_j \leftarrow 0}(\mathbf{b}_2) =$

$g|_{x_j \leftarrow 1}(\mathbf{b}_2)$ for all $\mathbf{b}_2 \in \mathbb{B}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = g|_{x_j \leftarrow 1}$, it follows that the ROBDD $g$ cannot be labelled with a variable $x_j$ because $g$ is reduced, hence a contradiction follows.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled $x_i$. Let $h = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = 1$ and $h|_{x_j \leftarrow 1}(\mathbf{b}_3) = 0$. Hence $g|_{x_i \leftarrow 0}$ contains a node $h$ labelled $x_j$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = 1$ and $h|_{x_j \leftarrow 1}(\mathbf{b}_3) = 0$. Hence $g$ contains a node labelled $x_j$ and $g$ is not labelled $x_i$ as required.

The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ case follows analogously. □

**Proposition 4.3.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_9$-symmetric in the pair $(x_i, x_j)$ and $i < j$ iff*

- *if whenever an ROBDD $g$ occurs in $f$ at a node labelled $x_i$ then every path through $g|_{x_i \leftarrow 0}$ visits a node $h$ labelled $x_j$ such that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$ and,*
- *every path from the root of $f$ that does not visit a node labelled $x_i$, visits a node $h$ labelled $x_j$ which satisfies the property that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$.*

Note how the wording of the second condition prevents a pair $(x_i, x_j)$ from being $T_9$-symmetric when there does not exist a node labelled $x_i$ nor $x_j$ in the ROBDD $f$. [1]

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n)$ for some $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$. Let $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Since $f$ is $T_9$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$. Thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = \neg h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ hence $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$ and it follows that the node $h$ labelled $x_j$ much be visited since an ROBDD is reduced.
- Arguing by the contrapositive, there are two cases to consider.

[1] We thank Laurent Mauborgne for alerting us to this oversight.

* Suppose there exists a path from the root of $f$ that does not visit a node labelled $x_i$ and does not visit a node labelled $x_j$. Thus $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ thus $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ hence $f$ is not $T_9$-symmetric in the variable pair $(x_i, x_j)$.
* Suppose there exists a path from the root of $f$ that does not visit a node labelled $x_i$ but visits a node $h$ labelled $x_j$ and $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$. Thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq \neg h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for some $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. There exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $b \in \mathbb{B}$ such that $h = f(\mathbf{b}_1, b, \mathbf{b}_2, x_j, \ldots, x_n)$. Therefore, $f(\mathbf{b}_1, b, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, b, \mathbf{b}_2, 1, \mathbf{b}_3)$. Since the path does not visit $x_i$, $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ hence $f$ is not $T_9$-symmetric in the variable pair $(x_i, x_j)$.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n)$. Then $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ thus $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$, hence condition 1 is violated irrespective of whether $h$ is visited.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ thus $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$, hence condition 2 is violated irrespective of whether $h$ is visited.

The $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$ case follows analogously. □

**Proposition 4.4.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_{10}$-symmetric in the pair $(x_i, x_j)$ and $i < j$ iff*

- *if whenever an ROBDD $g$ occurs in $f$ at a node labelled $x_i$ then every path through $g|_{x_i \leftarrow 1}$ visits a node $h$ labelled $x_j$ such that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$ and,*
- *every path from the root of $f$ that does not visit a node labelled $x_i$, visits a node $h$ labelled $x_j$ which satisfies the property that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$.*

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$. Let $h = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$ for some $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$. Let $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Since $f$ is $T_{10}$-symmetric in the variable

pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = \neg f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$. Thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = \neg h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ hence $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$ and it follows that the node $h$ labelled $x_j$ much be visited since an ROBDD is reduced.

- Arguing by the contrapositive, there are two cases to consider.
  - Suppose there exists a path from the root of $f$ that does not visit a node labelled $x_i$ and does not visit a node labelled $x_j$. Thus $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ thus $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$ hence $f$ is not $T_{10}$-symmetric in the variable pair $(x_i, x_j)$.
  - Suppose there exists a path from the root of $f$ that does not visit a node labelled $x_i$ but visits a node $h$ labelled $x_j$ and $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$. Thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq \neg h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for some $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. There exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $b \in \mathbb{B}$ such that $h = f(\mathbf{b}_1, b, \mathbf{b}_2, x_j, \ldots, x_n)$. Therefore, $f(\mathbf{b}_1, b, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, b, \mathbf{b}_2, 1, \mathbf{b}_3)$. Since the path does not visit $x_i$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$ hence $f$ is not $T_{10}$-symmetric in the variable pair $(x_i, x_j)$.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled $x_i$. Let $h = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Then $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ thus $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$, hence condition 1 is violated irrespective of whether $h$ is visited.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ thus $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$, hence condition 2 is violated irrespective of whether $h$ is visited.

The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$ case follows analogously. □

The first and second conditions of Propositions 4.1 and 4.2 can be checked in two depth-first traversals both requiring $O(n|G|)$ time and thus all $T_3$ and $T_4$-symmetries can be detected in $O(n|G|)$ time overall. Detecting $T_9$ and $T_{10}$-symmetries resides in $O(n^2|G|)$ since Propositions 4.3 and 4.4 imply that $T_9$ and $T_{10}$-asymmetries can be found by systematically searching through all pairs of variables $(x_i, x_j)$, checking that $f$ includes a path that neither contains $x_i$ nor $x_j$. These propositions assert that $T_3$, $T_4$, $T_9$ and $T_{10}$-symmetries are surprisingly tractable, and therefore suggest that these symmetries are particularly interesting for those applications where it is not necessary to compute all types of generalised symmetry [MM93, CM93a, ZCJMB04].

### 4.6.2 Slow Symmetries

Computing the remaining generalised symmetries, namely $T_2, T_5, T_6, T_7, T_8, T_{11}$ and $T_{12}$, requires more effort. The following four propositions explain how each of these symmetry relations can be computed in a series of passes where each pass computes all the symmetry types for each variable $x_i$.

**Proposition 4.5.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_2$-symmetric in the pair $(x_i, x_j)$ and $i < j$ iff*

- *every ROBDD rooted at a node labelled $x_i$ is $T_2$-symmetric in $(x_i, x_j)$ and,*
- *every path from the root to a node labelled $x_j$ passes through a node labelled $x_i$.*

*Proof.*

$\Leftarrow$ Consider the *if* direction.

- Since $f$ is $T_2$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ hence $g|_{x_i \leftarrow 0, x_j \leftarrow 0} = g|_{x_i \leftarrow 1, x_j \leftarrow 1}$.
- Suppose for the sake of a contradiction that there exists a path from the root to a node labelled $x_j$ that does not pass through a node labelled $x_i$. Thus, let $g = f(\mathbf{b}_1, 0, \mathbf{b}_2, x_j, \ldots, x_n) = f(\mathbf{b}_1, 1, \mathbf{b}_2, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$. Thus $g|_{x_j \leftarrow 0}(\mathbf{b}_3) = g|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for all $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = g|_{x_j \leftarrow 1}$ which is a contradiction since $g$ is reduced.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled $x_i$, thus there must exist some $\mathbf{b}_2$ and $\mathbf{b}_3$ such that $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 1, \mathbf{b}_3) = 0$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ as required.

The $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 1, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ case follows analogously. □

As before, Proposition 4.5 asserts that all $T_2$-symmetries can be found in two stages. The first stage, a lightweight preprocessing step, marks a pair $(x_i, x_j)$ as $T_2$-asymmetric if $f$ contains a path to a node labelled $x_j$ that does not pass through a node labelled

$x_i$. The second stage, which amounts to exhaustive search, searches for and examines each node labelled $x_i$ and checks whether the ROBDD rooted at that node is $T_2$-asymmetric in $(x_i, x_j)$. The first check is one of a number of checks carried out by the call to `GeneralRemoveAsymmetry` in the main loop of Algorithm 6. The second check is realised in the function `FindSlowAsymmetry` which precedes the main loop. `GeneralRemoveAsymmetry` and `FindSlowAsymmetry` also carry out checks to verify the first and second conditions of both Propositions 4.7 and 4.8. Note the simple structure of Proposition 4.6 permits $T_5$ and $T_6$ symmetries to be detected without a preprocessing step; these symmetries are solely detected within the `GeneralRemoveAsymmetry` procedure.

**Proposition 4.6.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_5$-symmetric (resp. $T_6$-symmetric) in the pair $(x_i, x_j)$ and $i < j$ iff every ROBDD rooted at a node labelled $x_i$ is $T_5$-symmetric (resp. $T_6$-symmetric) in $(x_i, x_j)$.*

*Proof.* For brevity, only the proof for the $T_5$ case is given.

- $\Leftarrow$ Consider the *if* direction. Suppose $g$ is an ROBDD rooted at a node labelled $x_i$. Then $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$. Since $f$ is $T_5$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3)$ for all $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$, thus $g|_{x_i \leftarrow 1, x_j \leftarrow 0} = g|_{x_i \leftarrow 0, x_j \leftarrow 0}$.

- $\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$. Suppose for the sake of a contradiction that $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$. Observe $g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$ and hence $g(0, \mathbf{b}_2, 0, \mathbf{b}_3) = g(1, \mathbf{b}_2, 0, \mathbf{b}_3)$ which is a contradiction. Therefore $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$ and $g$ is labelled by $x_i$. Hence there exists some $\mathbf{b}_2$ and $\mathbf{b}_3$ such that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 0, \mathbf{b}_3) = 0$ as required.
  The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ case follows analogously. □

**Proposition 4.7.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_7$-symmetric (resp. $T_8$-symmetric) in the pair $(x_i, x_j)$ and $i < j$ iff*

- *every ROBDD rooted at a node labelled $x_i$ is $T_7$-symmetric (resp. $T_8$-symmetric) in $(x_i, x_j)$ and,*
- *every path from the root of $f$ that does not visit a node labelled $x_i$ visits a node $h$ labelled $x_j$ which satisfies the property that $h|_{x_j \leftarrow 0} = \neg h|_{x_j \leftarrow 1}$.*

*Proof.* For brevity, only the proof for the $T_7$ case is given.

- $\Leftarrow$ Consider the *if* direction.

- Suppose $g$ is an ROBDD rooted at a node labelled $x_i$. Then $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$. Since $f$ is $T_7$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ for all $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$, hence $g|_{x_i \leftarrow 1, x_j \leftarrow 0} = \neg g|_{x_i \leftarrow 0, x_j \leftarrow 1}$.
- Arguing by the contrapositive, suppose there exists a path from the root of $f$ to a node $h$ labelled $x_j$ which does not visit a node labelled $x_i$ and $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$. Thus $h|_{x_j \leftarrow 0}(\mathbf{b}_3) \neq \neg h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ for some $\mathbf{b}_3 \in \mathbb{B}^{n-j}$. Let $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $a \in \mathbb{B}$ such that $h = f(\mathbf{b}_1, a, \mathbf{b}_2, x_j, \ldots, x_n)$. Therefore, $f(\mathbf{b}_1, a, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, a, \mathbf{b}_2, 1, \mathbf{b}_3)$. Since the path does not visit $x_i$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) \neq \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3)$ as required.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled by $x_i$. Thus there exists some $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 1, \mathbf{b}_3) = 1$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$.
  Hence $g$ is not labelled $x_i$. Let $h = g(0, \mathbf{b}_2, x_j, \ldots, x_n) = g(1, \mathbf{b}_2, x_j, \ldots, x_n)$. Observe $h|_{x_j \leftarrow 0} \neq \neg h|_{x_j \leftarrow 1}$ since $h|_{x_j \leftarrow 0}(\mathbf{b}_3) = h|_{x_j \leftarrow 1}(\mathbf{b}_3)$ as required.

The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 0$ case follows analogously. □

**Proposition 4.8.** *An ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_{11}$-symmetric (resp. $T_{12}$-symmetric) in the pair $(x_i, x_j)$ and $i < j$ iff*

- *every ROBDD rooted at a node labelled $x_i$ is $T_{11}$-symmetric (resp. $T_{12}$-symmetric) in $(x_i, x_j)$ and,*
- *every path from the root of $f$ passes through a node labelled $x_i$.*

*Proof.* For brevity, only the proof for the $T_{11}$ case is given.

$\Leftarrow$ Consider the *if* direction.

- Suppose $g$ is an ROBDD rooted at a node labelled $x_i$. Then $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$. Since $f$ is $T_{11}$-symmetric in the variable pair $(x_i, x_j)$, $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = \neg f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3)$ for all $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$, hence $g|_{x_i \leftarrow 1, x_j \leftarrow 0} = \neg g|_{x_i \leftarrow 0, x_j \leftarrow 0}$.
- Suppose for the sake of a contradiction that there exists a path from the root of $f$ to a node labelled $x_j$ which does not pass through a node labelled $x_i$. Let $g = f(\mathbf{b}_1, x_j, \ldots, x_n)$ for some $\mathbf{b}_1 \in \mathbb{B}^{j-1}$. Since $f|_{x_i \leftarrow 1, x_j \leftarrow 0} = \neg f|_{x_i \leftarrow 0, x_j \leftarrow 0}$

it follows that $g|_{x_j \leftarrow 0}(\mathbf{b}_2) = \neg g|_{x_j \leftarrow 0}(\mathbf{b}_2)$ for all $\mathbf{b}_2 \in \mathbb{B}^{n-j}$. Hence $g|_{x_j \leftarrow 0} = \neg g|_{x_j \leftarrow 0}$ which is a contradiction.

$\Rightarrow$ Consider the *only-if* direction, arguing by the contrapositive. Suppose there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{j-i-1}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j}$ such that $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$.

- Suppose $g|_{x_i \leftarrow 0} \neq g|_{x_i \leftarrow 1}$.
  Hence $g$ is labelled by $x_i$. Thus there exists some $\mathbf{b}_2$ and $\mathbf{b}_3$ such that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ and $g|_{x_i \leftarrow 0}(\mathbf{b}_2, 0, \mathbf{b}_3) = 1$ as required.
- Suppose $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$. Hence $g$ is not labelled $x_i$ as required.

The $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ and $f(\mathbf{b}_1, 0, \mathbf{b}_2, 0, \mathbf{b}_3) = 0$ case follows analogously. □

To increase the number of asymmetries discovered early in the execution of the algorithm and thus limit the cost of each call to `GeneralRemoveAsymmetry`, we first apply a number of lightweight sieves for the generalised symmetries. The following two lemmas detail structural properties of ROBDDs that hold in the presence of $T_5, T_6, T_7, T_8, T_{11}$ and $T_{12}$-symmetries. The absence of these properties imply that these symmetries cannot hold. In the case of Lemma 4.1, an algorithm in $O(n|G|)$ can be applied to ascertain whether every ROBDD rooted at a node labelled $x_i$ contains a node labelled $x_j$. This result therefore provides a sieve for $T_5$ and $T_6$-symmetries that can be incorporated into `FindSlowAsymmetry`. A sieve for $T_7, T_8, T_{11}$ and $T_{12}$-symmetries follows from Lemma 4.2 since the two cases of the lemma can both be checked in $O(n|G|)$ time. This is also implemented within `FindSlowAsymmetry`.

**Lemma 4.1.** *If an ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_5$-symmetric (resp. $T_6$-symmetric) in the pair $(x_i, x_j)$ and $i < j$ then every ROBDD rooted at a node labelled $x_i$ contains a node labelled $x_j$.*

*Proof.* For brevity, only the proof for the $T_5$ case is given. Suppose for the sake of a contradiction that there exists a node $g$ labelled $x_i$ that does not contain a node labelled $x_j$. Let $g = f(\mathbf{b}_1, x_i, \ldots, x_n)$ such that $g$ does not contain a node labelled $x_j$. Now $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = g|_{x_i \leftarrow 0}(\mathbf{b}_2, 0, \mathbf{b}_3)$ and since $g$ does not include $x_j$ it follows that $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 0, \mathbf{b}_3) = g|_{x_i \leftarrow 0}(\mathbf{b}_2, 1, \mathbf{b}_3)$ and $g|_{x_i \leftarrow 1}(\mathbf{b}_2, 1, \mathbf{b}_3) = g|_{x_i \leftarrow 0}(\mathbf{b}_2, 0, \mathbf{b}_3)$. Thus $g|_{x_i \leftarrow 1}(\mathbf{b}_2) = g|_{x_i \leftarrow 0}(\mathbf{b}_2)$ for all $\mathbf{b}_2 \in \mathbb{B}^{n-i}$. Hence $g|_{x_i \leftarrow 1} = g|_{x_i \leftarrow 0}$, it follows that the ROBDD $g$ cannot be labelled with a variable $x_i$ because $g$ is reduced, hence a contradiction follows. □

**Lemma 4.2.** *If an ROBDD $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_7$-symmetric (resp. $T_8$-symmetric, $T_{11}$-symmetric and $T_{12}$-symmetric) in the pair $(x_i, x_j)$ and $i < j$ then every ROBDD $g$ rooted at a node labelled $x_i$ satisfies the property that*

- *$g$ contains a node labelled $x_j$, or,*
- $g|_{x_i \leftarrow 0} = \neg g|_{x_i \leftarrow 1}$.

*Proof.* For brevity, only the proof for the $T_7$ case is given. To argue by contrapositive, suppose there exists a node $g$ labelled $x_i$ such that $g$ does not contain a node labelled $x_j$ and $g|_{x_i \leftarrow 0} \neq \neg g|_{x_i \leftarrow 1}$. Therefore $g|_{x_i \leftarrow 0}(\mathbf{b}_1, a, \mathbf{b}_2) = g|_{x_i \leftarrow 1}(\mathbf{b}_1, a, \mathbf{b}_2)$ for some $\mathbf{b}_1 \in \mathbb{B}^{j-i-1}$, $\mathbf{b}_2 \in \mathbb{B}^{n-j}$ and $a \in \mathbb{B}$. Since $g$ does not contain a node labelled $x_j$ it follows that $g|_{x_i \leftarrow 1}(\mathbf{b}_1, 0, \mathbf{b}_2) = g|_{x_i \leftarrow 0}(\mathbf{b}_1, 1, \mathbf{b}_2)$. Thus $f$ is $T_7$-asymmetric in the pair $(x_i, x_j)$ as required. □

The recursive structure of `GeneralRemoveAsymmetry` follows that of `RemoveAsymmetry` except that the call `GeneralRemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$ lies at its heart. `GeneralRemoveAsymmetryVar` in turn mimics the structure of `RemoveAsymmetryVar` except that it performs co-factor checks for $T_1, T_2, T_5, T_6, T_7, T_8, T_{11}$ and $T_{12}$-symmetries. Note that the $T_3, T_4, T_9$ and $T_{10}$-symmetries are already completely determined by `FindFastSymmetry` and hence need not be reconsidered. The complexity of a single call to `GeneralRemoveAsymmetryVar` is $O(|G|^2)$ and since this function can only be invoked a total of $|G|$ times from within Algorithm 6 it follows that the overall complexity of this procedure is $O(|G|^3)$. The preprocessing checks implemented within `FindSlowAsymmetry` for Propositions 4.1, 4.2, 4.5 and 4.8 all require $O(n|G|)$ time whereas the preprocessing required for Propositions 4.3, 4.4 and 4.7 take $O(n^2|G|)$. Algorithm 6 thus resides in $O(n^2|G| + |G|^3)$ overall.

**Algorithm 7** GeneralRemoveAsymmetry$(f, i, C)$

**Require:** $f \in ROBDD_X, i \in \mathbb{N} \cup \{0\}$ and $C \subset X$
1: **if** $C = \emptyset \vee f = true \vee f = false$ **then**
2: **return** $C$
3: **end if**
4: $j \leftarrow index(f)$
5: **if** $j > i$ **then**
6: **return** $C$
7: **else if** $j = i$ **then**
8: **return** `GeneralRemoveAsymmetryVar`$(f|_{x_i \leftarrow 0}, f|_{x_i \leftarrow 1}, C)$
9: **else**
10: $C \leftarrow$ `GeneralRemoveAsymmetry`$(f|_{x_j \leftarrow 0}, i, C)$
11: **return** `GeneralRemoveAsymmetry`$(f|_{x_j \leftarrow 1}, i, C)$
12: **end if**

### 4.6.3 Generalised Symmetry Propagation

To reduce the cost of each iteration of the main loop of Algorithm 6, one can apply asymmetry/symmetry propagation in the spirit of that employed in Algorithm 4. Tsai *et*

**Algorithm 8** GeneralRemoveAsymmetryVar$(g_0, g_1, C)$

**Require:** $g_0 \in ROBDD_X, g_1 \in \mathbb{B}_X$ and $C \subset X$

**if** $g_0 = true \vee g_0 = false$ **then**
  $j \leftarrow \infty$
**else**
  $j \leftarrow index(g_0)$
**end if**
**if** $g_1 = true \vee g_1 = false$ **then**
  $r \leftarrow \infty$
**else**
  $r \leftarrow index(g_1)$
**end if**
**if** $C = \emptyset \vee j = r = \infty$ **then**
  **return** $C$
**else if** $j = r$ **then**
  $(l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (j, g_0|_{x_j \leftarrow 0}, g_0|_{x_j \leftarrow 1}, g_1|_{x_r \leftarrow 0}, g_1|_{x_r \leftarrow 1})$
**else if** $j < r$ **then**
  $(l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (j, g_0|_{x_j \leftarrow 0}, g_0|_{x_j \leftarrow 1}, g_1, g_1)$
**else**
  $(l, g_{00}, g_{01}, g_{10}, g_{11}) \leftarrow (r, g_0, g_0, g_1|_{x_r \leftarrow 0}, g_1|_{x_r \leftarrow 1})$
**end if**
**if** $g_{10} \neq g_{01}$ **then**
  $C \leftarrow C \setminus \{(l, 1)\}$
**end if**
**if** $g_{00} \neq g_{11}$ **then**
  $C \leftarrow C \setminus \{(l, 2)\}$
**end if**
**if** $g_{00} \neq g_{10}$ **then**
  $C \leftarrow C \setminus \{(l, 5)\}$
**end if**
**if** $g_{01} \neq g_{11}$ **then**
  $C \leftarrow C \setminus \{(l, 6)\}$
**end if**
**if** $g_{10} \neq \neg g_{01}$ **then**
  $C \leftarrow C \setminus \{(l, 7)\}$
**end if**
**if** $g_{00} \neq \neg g_{11}$ **then**
  $C \leftarrow C \setminus \{(l, 8)\}$
**end if**
**if** $g_{00} \neq \neg g_{10}$ **then**
  $C \leftarrow C \setminus \{(l, 11)\}$
**end if**
**if** $g_{01} \neq \neg g_{11}$ **then**
  $C \leftarrow C \setminus \{(l, 12)\}$
**end if**
$C \leftarrow$ `GeneralRemoveAsymmetryVar`$(g_{00}, g_{10}, C)$
**return** `GeneralRemoveAsymmetryVar`$(g_{01}, g_{11}, C)$

*al.* [TM97] have reported transitivity results for some generalised symmetries, but to fully exploit asymmetry/symmetry propagation these results need to be extended to all 12 generalised symmetries. One such extension that involves $T_1$ and $T_3$-symmetries is presented in the following lemma:

**Lemma 4.3.** *If a Boolean function $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_1$-symmetric in the pair $(x_i, x_j)$ and $T_3$-symmetric in the pair $(x_j, x_k)$, then $f$ is $T_3$-symmetric in the pair $(x_i, x_k)$.*

*Proof.* Suppose ${T_1}^{x_i,x_j}(f)$ and ${T_3}^{x_j,x_k}(f)$ hold. Thus $f|_{x_i \leftarrow 1, x_j \leftarrow 0} = f|_{x_i \leftarrow 0, x_j \leftarrow 1}$, hence $f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 0, x_j \leftarrow 1, x_k \leftarrow 0}$ and $f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 1} = f|_{x_i \leftarrow 0, x_j \leftarrow 1, x_k \leftarrow 1}$. However $f|_{x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_j \leftarrow 0, x_k \leftarrow 1}$, hence we obtain $f|_{x_i \leftarrow 0, x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 0, x_j \leftarrow 0, x_k \leftarrow 1}$ and $f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 1}$. Therefore, $f|_{x_i \leftarrow 0, x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 0, x_j \leftarrow 0, x_k \leftarrow 1}$ and $f|_{x_i \leftarrow 0, x_j \leftarrow 1, x_k \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 1, x_j \leftarrow 0, x_k \leftarrow 1} = f|_{x_i \leftarrow 0, x_j \leftarrow 1, x_k \leftarrow 1}$. Hence $f|_{x_i \leftarrow 0, x_k \leftarrow 0} = f|_{x_i \leftarrow 0, x_k \leftarrow 1}$ and ${T_3}^{x_i,x_k}(f)$ holds. □

Table 7 summarises a collection of results that state implicational relationships between various generalised symmetries. For example, if ${T_3}^{x_i,x_j}(f)$ and ${T_4}^{x_j,x_k}(f)$ hold for some ROBDD $f$ then ${T_3}^{x_i,x_k}(f)$ also holds. Implicational relationships that have been previously reported [TM97] are marked with a †. Proofs for all the other implicational relationships of Table 7 can be found in the Appendix. Many of these results are established with proofs whose structure mirrors that used to substantiate lemma 4.3. The correctness of the remaining results flows from multiple applications of the following lemma that states equivalences between the generalised symmetries of the form ${T_i}^{x,y}(f)$ and ${T_j}^{y,x}(f)$ for any ROBDD $f$ for various $i, j \in \{1, \ldots, 12\}$.

**Lemma 4.4.**

- ${T_1}^{x,y}(f) \iff {T_1}^{y,x}(f)$ *and* ${T_7}^{x,y}(f) \iff {T_7}^{y,x}(f)$
- ${T_2}^{x,y}(f) \iff {T_2}^{y,x}(f)$ *and* ${T_8}^{x,y}(f) \iff {T_8}^{y,x}(f)$
- ${T_3}^{x,y}(f) \iff {T_5}^{y,x}(f)$ *and* ${T_9}^{x,y}(f) \iff {T_{11}}^{y,x}(f)$
- ${T_4}^{x,y}(f) \iff {T_6}^{y,x}(f)$ *and* ${T_{10}}^{x,y}(f) \iff {T_{12}}^{y,x}(f)$

*Proof.* For brevity only consider the positive cases.

- ${T_1}^{x,y}(f) \iff f|_{x \leftarrow 1, y \leftarrow 0} = f|_{x \leftarrow 0, y \leftarrow 1} \iff f|_{y \leftarrow 1, x \leftarrow 0} = f|_{y \leftarrow 0, x \leftarrow 1} \iff {T_1}^{y,x}(f)$
- ${T_2}^{x,y}(f) \iff f|_{x \leftarrow 0, y \leftarrow 0} = f|_{x \leftarrow 1, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 0} = f|_{y \leftarrow 1, x \leftarrow 1} \iff {T_2}^{y,x}(f)$
- ${T_3}^{x,y}(f) \iff f|_{x \leftarrow 0, y \leftarrow 0} = f|_{x \leftarrow 0, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 0} = f|_{y \leftarrow 1, x \leftarrow 0} \iff {T_5}^{y,x}(f)$
- ${T_4}^{x,y}(f) \iff f|_{x \leftarrow 1, y \leftarrow 0} = f|_{x \leftarrow 1, y \leftarrow 1} \iff f|_{y \leftarrow 0, x \leftarrow 1} = f|_{y \leftarrow 1, x \leftarrow 1} \iff {T_6}^{y,x}(f)$

□

The value of the above lemma is that it can be applied to show, for example, that the ${T_3}^{x,y}/{T_7}^{y,z}$ entry of Table II is a consequence of the ${T_7}^{x,y}/{T_5}^{y,z}$ entry. In fact three applications of the above lemma are needed to establish the correctness of the ${T_3}^{x,y}/{T_7}^{y,z}$ entry, as formalised in the following lemma.

**Lemma 4.5.** *If a Boolean function $f$ over a set of variables $\{x_1, \ldots, x_n\}$ is $T_3$-symmetric in the pair $(x, y)$ and $T_7$-symmetric in the pair $(y, z)$, then $f$ is $T_9$-symmetric in the pair $(x, z)$.*

*Proof.* Suppose ${T_3}^{x,y}(f)$ and ${T_7}^{y,z}(f)$ hold. By two applications of Lemma 4.4 it follows that ${T_5}^{y,x}(f)$ and ${T_7}^{z,y}(f)$ hold. Hence ${T_7}^{z,y}(f)$ and ${T_5}^{y,x}(f)$ hold. By Table 7 it follows that ${T_{11}}^{z,x}(f)$ holds and by another application of Lemma 4.4 it follows that ${T_9}^{x,z}(f)$ holds as required. □

We conjecture that no implicational symmetry relationships hold for the combinations of symmetry that lead to a blank entry in the table.

With the results of Table 7 in place, it is straightforward to construct an analogue of `SymmetryClosure`$(A, S)$ for generalised symmetries. The complexity of the generalised closure algorithm remains $O(n^3)$, assuming that an incremental algorithm is applied. Thus the overall running time of generalised symmetry detection with asymmetry/symmetry propagation is $O(n^3 + n^2|G| + |G|^3)$.

## 4.7 Experimental Results

Tables 8 and 9 presents a comparison between the generalised symmetry algorithm of Zhang *et al.* [ZCJMB04] and the generalised anytime approach with variable sifting enabled and variable sifting disabled, respectively. Mishchenko's implementation of his own algorithm was modified to detect $T_1, T_2, T_7$ and $T_8$-symmetries following the ideas prescribed by Zhang *et al.* The timings given for the anytime algorithm presented in this chapter reflect the time required to compute all 12 generalised symmetry types. Note therefore, that the anytime algorithm is inferring 8 generalised symmetries which are not computed by Zhang *et al.* This algorithm applies asymmetry/symmetry propagation between iterations of the main loop and uses all sieves described thus far. As before, enabling garbage collection has no discernible impact on the running time of the anytime algorithm. Although the generalised algorithm is not consistently faster than that of Zhang *et al*, it is nevertheless attractive because the runtime never exceeds `7200` seconds for any benchmarks, and for many benchmarks, it is significantly faster. For completeness, Table 10 summarises the results of timing experiments performed with an Intel Core2 Duo 2.33GHZ PC (using just one core), equipped with 2GB of RAM, running MacOSX.

## 4.8 Conclusions

This chapter demonstrates that anytime symmetry detection techniques extend naturally from classical to generalised symmetries. The generalised algorithm retains a multipass structure thereby permitting sieves and transitivity to be exploited in order to accelerate symmetry detection. The sieves not only speedup symmetry detection but also allow symmetry detection to be decomposed into series of passes without compromising correctness. The sieves are interesting within themselves since the $T_3, T_4, T_9$ and $T_{10}$ symmetries can be completely discovered without employing search; the tractability of these symmetries makes them particularly suitable for permutation independent Boolean comparison. Furthermore, the transitivity results also have applications outside ROBDD manipulation. In terms of generality, the anytime generalised symmetry detection algorithm presented in this chapter is capable of detecting all 12 symmetry types. In terms of efficiency, the performance of the algorithm compares very favourably with that of Zhang for time whilst requiring negligible space.

With a view to the future, the iterative nature of the anytime algorithms make them good candidates for parallel evaluation on the 8 and 16 core processors that are predicated to emerge over the next 5 years. Although the speedups achieved by parallel evaluation of BDD operations have often been modest [MJH98], the weak coupling between the iterations of the main loop of the symmetry detection algorithms – the property that yields to anytime execution – also leads to weakly coupled parallel execution.

| | $T_1^{y,z}$ | $T_7^{y,z}$ | $T_2^{y,z}$ | $T_8^{y,z}$ | $T_3^{y,z}$ | $T_9^{y,z}$ | $T_4^{y,z}$ | $T_{10}^{y,z}$ | $T_5^{y,z}$ | $T_{11}^{y,z}$ | $T_6^{y,z}$ | $T_{12}^{y,z}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $T_1^{x,y}$ | $T_1^{x,z}$ † | $T_7^{x,z}$ † | $T_2^{x,z}$ † | $T_8^{x,z}$ † | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_5^{x,z}$ | $T_{11}^{x,z}$ | $T_6^{x,z}$ | $T_{12}^{x,z}$ |
| $T_7^{x,y}$ | $T_7^{x,z}$ † | $T_1^{x,z}$ † | $T_8^{x,z}$ † | $T_2^{x,z}$ † | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_{11}^{x,z}$ | $T_5^{x,z}$ | $T_{12}^{x,z}$ | $T_6^{x,z}$ |
| $T_2^{x,y}$ | $T_2^{x,z}$ † | $T_8^{x,z}$ † | $T_1^{x,z}$ † | $T_7^{x,z}$ † | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_5^{x,z}$ | $T_{11}^{x,z}$ | $T_6^{x,z}$ | $T_{12}^{x,z}$ |
| $T_8^{x,y}$ | $T_8^{x,z}$ † | $T_2^{x,z}$ † | $T_7^{x,z}$ † | $T_1^{x,z}$ † | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_{11}^{x,z}$ | $T_5^{x,z}$ | $T_{12}^{x,z}$ | $T_6^{x,z}$ |
| $T_3^{x,y}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | | | | |
| $T_9^{x,y}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | $T_3^{x,z}$ | $T_9^{x,z}$ | | | | |
| $T_4^{x,y}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | | | | |
| $T_{10}^{x,y}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | $T_4^{x,z}$ | $T_{10}^{x,z}$ | | | | |
| $T_5^{x,y}$ | $T_5^{x,z}$ | $T_5^{x,z}$ | $T_6^{x,z}$ | $T_6^{x,z}$ | | | | | $T_5^{x,z}$ | $T_5^{x,z}$ | $T_6^{x,z}$ | $T_6^{x,z}$ |
| $T_{11}^{x,y}$ | $T_{11}^{x,z}$ | $T_{11}^{x,z}$ | $T_{12}^{x,z}$ | $T_{12}^{x,z}$ | | | | | $T_{11}^{x,z}$ | $T_{11}^{x,z}$ | $T_{12}^{x,z}$ | $T_{12}^{x,z}$ |
| $T_6^{x,y}$ | $T_6^{x,z}$ | $T_6^{x,z}$ | $T_5^{x,z}$ | $T_5^{x,z}$ | | | | | $T_5^{x,z}$ | $T_5^{x,z}$ | $T_6^{x,z}$ | $T_6^{x,z}$ |
| $T_{12}^{x,y}$ | $T_{12}^{x,z}$ | $T_{12}^{x,z}$ | $T_{11}^{x,z}$ | $T_{11}^{x,z}$ | | | | | $T_{11}^{x,z}$ | $T_{11}^{x,z}$ | $T_{12}^{x,z}$ | $T_{12}^{x,z}$ |

Table 7: Transitivity results

| Circuit | # In | # Out | Σ\|G\| | \|S\| | Read | Naïve | Zhang-GC | Generalised |
|---|---|---|---|---|---|---|---|---|
| pair | 173 | 137 | 118066 | 15949 | 0.20 | 219.76 | 64.27 | 9.10 |
| C3540 | 50 | 22 | 4618194 | 1892 | 21.80 | >7200 | 162.91 | 186.32 |
| C880 | 60 | 26 | 600998 | 1759 | 8.29 | 1309.88 | 42.39 | 44.52 |
| s4863 | 153 | 104 | 126988 | 3825 | 2.63 | 33.10 | 15.20 | 5.42 |
| s9234.1 | 247 | 250 | 4434504 | 22410 | 20.14 | >7200 | >7200 | 287.62 |
| s38584.1 | 1464 | 1730 | 150554 | 136537 | 3.70 | 501.34 | 576.39 | 10.30 |
| simp10 | 105 | 1 | 722074 | 133 | 58.45 | >7200 | 685.70 | 233.45 |
| simp12 | 117 | 1 | 758330 | 135 | 76.23 | >7200 | >7200 | 304.21 |
| simp14 | 120 | 1 | 562326 | 137 | 70.38 | >7200 | 1114.29 | 75.75 |
| hom06 | 104 | 1 | 1176845 | 236 | 65.22 | >7200 | 445.78 | 367.89 |
| hom08 | 95 | 1 | 893312 | 108 | 56.48 | >7200 | 482.30 | 281.57 |
| hom10 | 130 | 1 | 309221 | 180 | 29.98 | >7200 | 1510.32 | 35.85 |
| ca004 | 53 | 1 | 782640 | 27 | 5.40 | >7200 | 234.12 | 73.10 |
| ca008 | 96 | 1 | 682617 | 110 | 20.40 | >7200 | 439.59 | 235.86 |
| ca016 | 107 | 1 | 861209 | 147 | 60.10 | >7200 | 305.11 | 72.68 |
| urquhart2_25 | 48 | 1 | 722657 | 34 | 3.06 | >7200 | 125.36 | 91.80 |
| urquhart3_25 | 62 | 1 | 1771025 | 162 | 6.22 | >7200 | >7200 | 293.45 |
| urquhart4_25 | 68 | 1 | 1736705 | 184 | 5.96 | >7200 | >7200 | 83.44 |
| rope_0002 | 54 | 1 | 634914 | 52 | 3.06 | >7200 | 482.34 | 81.68 |
| rope_0004 | 62 | 1 | 1052214 | 76 | 4.73 | >7200 | 1087.19 | 186.29 |
| rope_0006 | 61 | 1 | 759039 | 76 | 3.14 | >7200 | 657.74 | 35.78 |
| ferry8 | 111 | 1 | 290127 | 158 | 78.35 | >7200 | 174.12 | 176.22 |
| ferry10 | 116 | 1 | 539419 | 174 | 88.08 | >7200 | 3146.64 | 365.93 |
| ferry12 | 123 | 1 | 277291 | 217 | 47.96 | >7200 | 142.10 | 37.63 |
| gripper10 | 125 | 1 | 393485 | 188 | 69.08 | >7200 | 528.66 | 295.67 |
| gripper12 | 129 | 1 | 667877 | 220 | 50.95 | >7200 | 673.09 | 587.28 |
| gripper14 | 118 | 1 | 767735 | 173 | 47.29 | >7200 | 804.21 | 621.99 |

Table 8: Generalised Symmetry Experimental Results without Sifting

| Circuit | # In | # Out | Σ\|G\| | \|S\| | Read | Naïve | Zhang-GC | Generalised |
|---|---|---|---|---|---|---|---|---|
| pair | 173 | 137 | 8599 | 15949 | 0.60 | 4.56 | 1.53 | 1.21 |
| C3540 | 50 | 22 | 43334 | 1892 | 14.00 | 72.74 | 5.47 | 5.43 |
| C880 | 60 | 26 | 8753 | 1759 | 0.44 | 9.67 | 0.62 | 1.13 |
| s4863 | 153 | 104 | 75549 | 3825 | 87.58 | 25.25 | 14.20 | 4.36 |
| s9234.1 | 247 | 250 | 9376 | 22410 | 2.16 | 13.53 | 3.78 | 1.12 |
| s38584.1 | 1464 | 1730 | 34833 | 136537 | 13.10 | 30.44 | 246.37 | 2.59 |
| simp10 | 105 | 1 | 222431 | 133 | 205.11 | >7200 | 72.62 | 140.11 |
| simp12 | 117 | 1 | 292811 | 135 | 230.61 | >7200 | 70.33 | 202.89 |
| simp14 | 120 | 1 | 86267 | 137 | 111.84 | >7200 | 163.74 | 63.42 |
| hom06 | 104 | 1 | 60357 | 236 | 170.76 | >7200 | 71.94 | 58.78 |
| hom08 | 95 | 1 | 110160 | 108 | 128.91 | >7200 | 71.44 | 113.58 |
| hom10 | 130 | 1 | 142827 | 180 | 283.80 | >7200 | 315.23 | 138.62 |
| ca004 | 53 | 1 | 9119 | 27 | 1.86 | 26.92 | 3.46 | 1.75 |
| ca008 | 96 | 1 | 19945 | 110 | 3.80 | 569.14 | 207.31 | 12.50 |
| ca016 | 107 | 1 | 90033 | 147 | 33.45 | >7200 | 198.45 | 10.78 |
| urquhart2_25 | 48 | 1 | 41098 | 34 | 6.86 | 227.51 | 0.96 | 14.02 |
| urquhart3_25 | 62 | 1 | 43599 | 162 | 3.03 | 1832.21 | >7200 | 32.61 |
| urquhart4_25 | 68 | 1 | 45008 | 184 | 23.21 | >7200 | >7200 | 67.70 |
| rope_0002 | 54 | 1 | 1038 | 52 | 0.15 | 2.82 | 0.34 | 0.17 |
| rope_0004 | 62 | 1 | 11874 | 76 | 2.29 | 246.93 | 64.10 | 9.30 |
| rope_0006 | 61 | 1 | 11066 | 76 | 5.01 | 781.21 | 17.20 | 14.93 |
| ferry8 | 111 | 1 | 5998 | 158 | 22.10 | >7200 | 156.76 | 30.65 |
| ferry10 | 116 | 1 | 3141 | 174 | 6.18 | 210.82 | 3050.82 | 3.91 |
| ferry12 | 123 | 1 | 3758 | 217 | 21.18 | 1145.56 | 191.93 | 14.26 |
| gripper10 | 125 | 1 | 17525 | 188 | 183.67 | >7200 | 68.67 | 224.31 |
| gripper12 | 129 | 1 | 17035 | 220 | 165.65 | >7200 | 59.98 | 247.64 |
| gripper14 | 118 | 1 | 9742 | 173 | 160.23 | >7200 | 63.40 | 185.57 |

Table 9: Generalised Symmetry Experimental Results with Sifting

| | with reordering | | | without reordering | | |
|---|---|---|---|---|---|---|
| **Circuit** | **Naïve** | **Zhang** | **Close** | **Naïve** | **Zhang** | **Close** |
| alu2 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| alu4 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| C1908 | 3.75 | 0.17 | 1.23 | 5.50 | 0.29 | 1.16 |
| C2670 | 53.46 | 2.58 | 2.62 | – | – | – |
| C3540 | 24.37 | 3.29 | 1.59 | – | – | – |
| C432 | 0.35 | 0.01 | 0.02 | 10.23 | 15.70 | 0.88 |
| C499 | 32.18 | 0.08 | 1.02 | 40.80 | 0.33 | 3.23 |
| C5315 | 4.71 | 0.40 | 0.62 | – | – | – |
| C880 | 2.52 | 0.35 | 0.34 | 392.50 | 628.10 | 19.35 |
| dalu | 0.58 | 0.05 | 0.06 | 0.77 | 0.14 | 0.32 |
| des | 0.24 | 0.19 | 0.15 | 0.44 | 0.60 | 0.25 |
| frg2 | 0.13 | 0.08 | 0.07 | 0.27 | 0.19 | 0.11 |
| i10 | 58.68 | 115.71 | 6.96 | >7200 | 4556.79 | 20.81 |
| k2 | 0.50 | 0.08 | 0.05 | 0.50 | 0.08 | 0.05 |
| pair | 1.67 | 0.38 | 0.40 | 73.58 | 15.54 | 2.96 |
| rot | 1.70 | 0.18 | 0.23 | 9.02 | 2.40 | 0.86 |
| s635 | 0.05 | 0.04 | 0.04 | 0.05 | 0.04 | 0.04 |
| s838.1 | 0.10 | 0.04 | 0.05 | 0.12 | 0.04 | 0.05 |
| s1196 | 0.07 | 0.03 | 0.02 | 0.15 | 0.04 | 0.03 |
| s1269 | 0.39 | 0.07 | 0.06 | 0.60 | 0.10 | 0.10 |
| s1423 | 1.60 | 0.21 | 0.30 | 8.96 | 0.80 | 0.81 |
| s3271 | 0.06 | 0.05 | 0.04 | 0.82 | 0.18 | 0.13 |
| s4863 | 8.45 | 0.41 | 0.96 | 17.48 | 1.23 | 1.75 |
| s9234.1 | 1.89 | 0.50 | 0.51 | – | – | – |
| too_large | 0.58 | 0.10 | 0.06 | 0.56 | 0.09 | 0.03 |

Table 10: Generalised Symmetry Timing Experiments on an Intel

# Chapter 5

# Widening ROBDDs with Prime Implicants

"The optimist thinks this is the best of all possible worlds.
The pessimist fears it is true."
- J. R. Oppenheimer

**Abstract.** In this chapter we present an anytime algorithm for ROBDD approximation. The need for approximation arises because many ROBDD operations result in an ROBDD whose size is quadratic in the number of nodes in the inputs. When ROBDDs are applied in abstract interpretation, the additional problem arises that monotonically increasing chains of ROBDDs can be computed whose length is exponential in the number of variables. This chapter proposes an approximation technique, a widening, that can be used to both constrain the size of an ROBDD and also bound chain length by some given constant. The widening can be used to either systematically approximate from above (i.e. derive an ROBDD of a weaker Boolean function) or below (i.e. infer a stronger Boolean function).

## 5.1 Introduction

The popularity of ROBDDs in model checking [BCM+92], program analysis [WL04] and abstract interpretation [BS99] stems from their apparent memory-efficient encoding of Boolean functions and a canonical representation that supports the memoisation of ROBDD operations. However, the inherent intractability of Boolean function manipulation inevitably manifests itself; even though ROBDDs are *reduced*, that is, ROBDDs are constructed so as to factor out all isomorphic sub-ROBDDs. Since the problem of testing satisfiability is NP-complete [Coo71] and a satisfiability check can be performed

in constant time for an ROBDD it follows that the computational effort required to construct an ROBDD representing an arbitrary Boolean function is worst-case exponential in the number of variables, unless P = NP. Furthermore, Boolean functions exist whose ROBDD representation requires a number of nodes that is exponential in the number of variables irrespective of the variable ordering employed [Bry91].

To combat these insurmountable complexity results, we have one of two choices. One option is to rely upon ROBDD minimisation techniques to obtain a suitably small ROBDD representation. However, the results of Wegener [Weg00] and Sieling [Sie02] provide strong indications as to the limitations of minimisation: Wegener [Weg00] proves that computing an optimal variable ordering is itself NP-complete; Sieling [Sie02] provides the accompanying result that it is not possible in polynomial time to compute a variable ordering that results in an ROBDD whose size is within a given constant factor $\epsilon$ of the optimal size. Furthermore, minimisation techniques often need to be repeatedly applied in applications such as model checking and abstract interpretation since they are themselves repetitive, iterative processes. In fact ROBDD packages such as CUDD [Som05] can apply minimisation many times whilst computing the result of even a single ROBDD operation.

An alternative to minimisation is to apply approximation techniques, which in some applications, provides a way to improve the space and time behaviour of an analysis without compromising correctness [RS95, Shi96, Fec97, Mau98, RMSS98]. In the context of ROBDDs, the approximation problem is that of formulating an algorithm which takes, as input, an ROBDD $g$ of a Boolean function $f$ and computes, as output, an ROBDD $g'$ of a Boolean function $f'$ such that $f \models f'$ and $|g'| \leq |g|$. This problem statement stipulates the correctness criterion for ROBDD approximation. However, the statement is somewhat incomplete in that it permits the trivial solution *true* and $g$ to be generated for a given ROBDD $g$ since the former can be represented in exactly 1 node and the latter in $|g|$ nodes. In the context of a particular application, an approximation $g'$ such that $|g'| \approx \frac{|g|}{2}$ is likely to be far more useful than the vacuous approximations *true* and $g$. The challenge therefore, is to derive a principled way for computing an ROBDD approximation which achieves a suitable reduction in ROBDD size.

## 5.2 Applications

### 5.2.1 Program Analysis and Large ROBDDs

Although Cook's iconic theorem [Coo71] is usually interpreted as a statement about Boolean satisfiability, the proof of the result is an interesting comment on the expressive power of Boolean functions. The proof shows how the behaviour of any Turing machine can be modelled by a Boolean formulae. As a consequence, it is evident that Boolean formulae constitute a computational domain that can, at least in theory, be used to

reason about any program without any loss of precision.

More usually, Boolean formulae arise in program analysis through the use of abstraction. Such analyses typically manipulate propositional formulae defined over a given finite set of propositional variables $x_1, \ldots, x_n$. Each propositional variable is used to indicate whether the state possesses a property of interest. A formula such as $x_1 \rightarrow x_3$ is then interpreted as stating that whenever the property for $x_1$ holds then it follows that $x_3$ holds also. This approach has been applied in suspension analysis [GK08], pair-sharing analysis [LS02], groundness analysis [AMSS98], strictness analysis [SR95], finiteness analysis [BGHZ04] and set-sharing analysis [CSS99].

Due to the intractability of modelling programs with propositional formulae containing loops and recursion, the usual tactic is to unroll each loop to a predetermined depth $k$ [JV00]. If no fault can be found in the program, then the analysis is reapplied to depth $k+1$, $k+2$, etc. until the analysis exhausts the available resources. The value of this technique is that it provides a way to manage the size of formulae. Even when abstraction is applied, Boolean functions can [Cod99] and do [Fec97] arise whose ROBDD representation is intractably large. Furthermore, it is well-known that the domain of Positive Boolean functions [AMSS98] (**Pos**) contains formulae whose ROBDD representation requires exponential space [Cod99]. For instance, when an analysis associates a program variable with $n$ attributes and $m$ such program variables appear in scope, then an ROBDD over $m\lceil\log(n)\rceil$ propositional variables is required to encode the dependencies between the attributes of the program variables [GHB05]. Even when tree-automata techniques are used to improve the encoding, problematically large ROBDDs can arise [GHB05].

### 5.2.2 Abstract Interpretation and Long Chains

In iterative fixed-point based analyses, such as those formulated with abstract interpretation, the tractability problems are further compounded by the fact that ROBDDs are not only problematic in terms of space, but also in terms of time. This is not only due to the complexity of individual ROBDD operations, but because the number of ROBDD operations is itself potentially exponential. Suppose, for example, that the result of an analysis is conceived as the least fixed-point of a series of Boolean equations:

$$\begin{array}{lcl} f_1 & = & F_1(f_1, \ldots, f_k) \\ \vdots & \vdots & \vdots \\ f_k & = & F_k(f_1, \ldots, f_k) \end{array}$$

where each $f_i \in \mathbb{B}^X$ is a Boolean function over a set of variables $X = \{x_1, \ldots, x_n\}$ and each $F_i$ is an operation on $f_1, \ldots, f_k$ obtained by, say, composing monotonic Boolean operations such as disjunction $f_i \vee f_j$, conjunction $f_i \wedge f_j$ and existential quantification

$\exists_{x_i}(f_j)$. The least fixed-point of the series of $k$ Boolean equations $f_1, \ldots, f_k$ can be computed by setting $f_i = \mathit{false}$ and then reapplying the $k$ equations until stability is achieved. In the worst-case, each application of $k$ equations might weaken exactly one function $f_i$ by adding a single model. Since each $f_i$ can possess a maximum of $2^n$ models, a chain of $k \cdot 2^n$ iterates are required in the worst-case which is clearly exponential of order $n$ and thus violates the requirement for a polynomial analysis which is usually considered necessary for tractability. (The reader is referred to [Cod99] for examples that manifest this behaviour.)

With the advent of Quantum computers it may well be possible to perform an exponential number of iterations in only a polynomial number of operations. Without access to such machines, it is usually considered better to compute an approximate answer to a fixed-point calculation in an acceptable time than an exact answer in an exorbitant time. To this end, widening operators have been proposed [CC77, CC92b] that accelerate convergence on domains that possess either infinite or very long chains. In the context of ROBDDs a widening is a function $\nabla \in \mathbb{B}_X \times \mathbb{B}_X \to \mathbb{B}_X$ such that,

- $x \models x \nabla y$ holds for all $x, y \in \mathbb{B}_X$ and,
- $y \models x \nabla y$ holds for all $x, y \in \mathbb{B}_X$ and,
- for all increasing chains $x^0 \models x^1 \models \ldots$, the increasing chain defined by $y^0 = x^0, \ldots, y^{i+1} = y^i \nabla x^{i+1}, \ldots$ is not strictly increasing.

Although the above formulation of a widening is sufficient to guarantee termination [CC77, CC92b], more aggressive approximation may be necessary to compute a fixed-point in an acceptable amount of time. Rather surprisingly, despite extensive literature on reducing the size of an ROBDD by selecting a propitious variable ordering, the problem of accurately widening ROBDDs for time has received relatively scant attention.

### 5.2.3 Constraint Programming

The fundamental idea in constraint programming is that constraints can be used to represent a problem, solve it, and represent the solution [Apt03]. Constraint Satisfaction Problems (`CSPs`) represent one of the most widely occurring classes of constraint problems. An instance of the `CSP` problem consists of a set of variables $X$, each of which corresponds to a single element of a discrete and finite set $D$, and a finite set of constraints $\Sigma$ between the variables that specify which combinations of values are permissible and which are not. Formally, a `CSP` instance is a triple $\langle X, D, \Sigma \rangle$ where $X$ is a set of $n$ variables, $D$ is a finite domain of values, and $\Sigma$ is a finite set of constraints. Every constraint $\sigma \in \Sigma$ is in turn a pair $\langle \mathbf{x}, R \rangle$ where $\mathbf{x} \in X^*$ is a vector of variables and $R \subseteq D^m$ where $|\mathbf{x}| = m \leq n$. An evaluation of the variables satisfies a constraint $\langle \langle x_1, \ldots, x_n \rangle, R \rangle$ iff $\langle v(x_1), \ldots, v(x_n) \rangle \in R$ for some function $v : X \to D$. A solution

to the CSP is an evaluation that satisfies all constraints. Solving a CSP instance is NP-complete [Coo71] and if $D$ is relaxed to be infinite, the satisfaction problem becomes undecidable [Dav82].

The problem of widening ROBDDs is pertinent to CSP due to an encoding proposed by Lagoon *et al.* [LS04, HLS04, HLS05]. The encoding explains how an arbitrary CSP $\langle X, D, \Sigma\rangle$ may be translated into a binary CSP $\langle X', \mathbb{B}, \Sigma'\rangle$. Formally, the encoding $e$ is a bijective map with signature $e : (X \to D) \to (X' \to \mathbb{B})$ such that $v$ is a solution of $\langle X, D, \Sigma\rangle$ if and only if $e(v)$ is a solution of the binary CSP $\langle X', \mathbb{B}, \Sigma'\rangle$. The value of this encoding is that groups of constraints, for instance, $\sigma_1, \sigma_2, \sigma_3 \in \Sigma$, can be encoded as a single ROBDD which improves the propagation between these three constraints and thereby speeds up the constraint solving process [LS04, HLS04, HLS05]. However, the value of this approach is compromised by the size of the resulting ROBDDs. This suggests approximating an ROBDD with another of smaller size so as to maintain the computational advantage of this approach even when a system of constraints results in an infeasibly large ROBDD. If the ROBDD were over-approximated, then it would be necessary to use the ROBDD in conjunction with the original constraints; if the ROBDD were under-approximated, then the original constraints can be removed though the possibility arises that some solutions to the original CSP may be discarded.

### 5.2.4 Reachability Analysis

ROBDD approximation arises in reachability analysis [RS95] in which ROBDDs are used to represent finite, albeit, very large sets of states. To illustrate the basic ideas, let $\Sigma$ denote a finite set of states, and $t \subseteq \Sigma \times \Sigma$ denote a transition system. The transition system can be lifted to sets of states by $T(S) = \{s' \mid s' \in S \wedge t(s, s')\}$ where $S \subseteq \Sigma$. The set of states reachable from an initial set of states $S_0$ is $\bigcup_{i=1}^{\infty} S_i$ where $S_{i+1} = T(S_i)$. Since the set of states $S$ is finite, it is possible to represent the set of states $S_i$ as an ROBDD $f_i$ and $T$ as an operation on ROBDDs. The problem of set reachability then amounts to computing $\bigvee_{i=1} f_i$ [CGP00]. Even then the ROBDDs $f_i$ can become unmanageably large, which motivates over-approximating the intermediary ROBDDs $f_i$ and thus the result $\bigvee_{i=1} f_i$. Reachability questions can still be answered in the negative, that is, it is possible to infer that a state $s'$ cannot be reached from $S_0$.

## 5.3 Contributions

In this chapter we propose a new widening for Boolean functions represented as ROBDDs based upon the enumeration of prime implicants [CM92a] that has a number of attractive properties:

- The widening can be used to accelerate fixed-point computation by ensuring that a Boolean function $f$ can be weakened no more than a prescribed number of times.

This makes the widening particularly useful in abstract interpretation based iterative program analysis and model checking applications requiring fixed-point computations. Previous attempts at bounding the number of iterations have resorted to confining the application to a fixed sub-domain of Boolean formulae [HACK00]. The widening presented in this chapter can support far richer classes of dependencies without sacrificing scalability.

- The widening can compute *dense* approximations of an ROBDD. The density of an ROBDD $g$ is defined as the ratio of the number of minterms, $|model_X(f)|$, in the represented function $f$, to the number of nodes, $|g|$, in the representing ROBDD. Moreover, by constructing the new ROBDD in terms of the progressively longer implicants, the widening can be tuned to achieve the desired tradeoff between precision and density in an anytime fashion.

- The widening is not dependent on the variable ordering, that is, the approximations obtained are invariant with respect to the variable ordering employed. State-of-the-art in ROBDD approximation is represented by heuristic algorithms [Shi96, RMSS98] that prune branches from an ROBDD by checking whether each branch is subsumed by its sibling. These algorithms are syntactic in that they are informed only by the structure of the ROBDD to be approximated. The widening presented herein is formulated in terms of the prime implicants of the underlying Boolean function, a property which is invariant with respect to representation. The advantage of this semantic approach is that the widening is not sensitive to the variable ordering, hence improving the predictability of an application utilising the widening.

- The widening can be realised in a surprisingly straightforward manner by introducing a cardinality constraint into the algorithm of Coudert and Madre [CM92a] that removes all prime implicants of excessive length. Experimental work suggests that although this widening produces accurate approximations, the running time of the implementation is not significantly worse than state-of-the-art methods [Shi96, RMSS98].

The remainder of this chapter is structured as follows: Section 5.4 surveys the related work. Section 5.5 specifies a widening for ROBDDs and §5.6 details algorithms for realising it. Section 5.7 presents the experimental results. Section 5.8 presents a widening based on anti-unification [Plo70] that is provably polynomial both in space and time that offers a compromise between efficiency and generality that may suit some analyses [HACK00]. Finally, §5.9 concludes.

## 5.4 Related Work

### 5.4.1 Widening

The use of widening operators [CC77, CC92b] is prevalent in abstract interpretation. Despite a lack of work on widening the domain of Boolean formulae represented as ROBDDs, many widenings have been proposed for a multitude of computational domains such as intervals [CC92b] and polyhedra [BHRZ05, GH06]. However, several algorithms have been proposed for approximating ROBDDs [RS95, Shi96, RMSS98], although they are *ad hoc* in their approach, and their results are somewhat unpredictable. Nevertheless, the approximations they deliver, provide a useful basis for comparison. The ROBDD approximation algorithms of Ravi and Somenzi [RS95], Shiple [Shi96] and Ravi *et al* [RMSS98] seek to compute an over-approximation of an ROBDD such that the resultant approximation possesses a significant improvement in density which henceforth will be denoted by $\delta(g)$ for an ROBDD $g$ [RS95].

The simplest of these heuristic algorithms are the *Heavy-branch sub-setting* (HB) and *Short-path sub-setting* (SP) algorithms of Ravi and Somenzi [RS95], both of which deliver under-approximations rather than over-approximations. Both algorithms comprise a two-pass procedure. The HB algorithm computes in the first pass, the number of minterms in the sub-ROBDD rooted at each node in the input ROBDD and the number of nodes that would be eliminated by replacing the sub-ROBDD with the Boolean constant *false*. The second pass proceeds from the root of the ROBDD replacing the join edge of the so-called lightest branch [RS95], that is, the child with fewest minterms, with the Boolean constant *false*. The process continues until the size of the remainder ROBDD satisfies a predefined threshold. The resultant ROBDD then contains a set of nodes upper-most in the variable order, each with one child as the constant *false* [RMSS98]. Hence the algorithm has the effect of biasing the approximation to the variables that are upper-most in the variable ordering.

The SP algorithm is based on the idea that short paths in an ROBDD correspond to implicants that are defined over a small number of variables yet contribute a large number of truth assignments. The algorithm proceeds in its first pass, by determining the length of the shortest paths through each node. In the second pass, a sub-ROBDD is replaced with the Boolean constant *false* whenever the length of the shortest path through the node that roots the sub-ROBDD exceeds a prescribed threshold. Both the HB and SP algorithms reside in $O(|G|)$ which as Ravi *at al.* concede, "permits only a limited study of the ROBDD under consideration" [RMSS98].

Shiple [Shi96] and Ravi *et al.* [RMSS98] attempt to refine the HB and SP algorithms by applying further heuristics. These two works propose the algorithms that are referred to as `bddOverApprox` and `remapOverApprox` respectively in the Colorado University Decision Diagram (CUDD) package [Som05]. These algorithms employ a more thorough

inspection of the ROBDD, and hence both reside in $O(|G|^2)$. The `bddOverApprox` procedure computes for each node a lower bound on the increase in density that would follow from its replacement [RMSS98] where nodes are then replaced using either the HB or SP strategies. The `remapOverApprox` algorithm additionally permits the remapping of a node $g$ to another node $g'$ of the ROBDD by redirecting an edge that leads to $g$ to $g'$. Replacing nodes in this manner increases sharing producing a more dense ROBDD representation whilst attaining greater accuracy. In both algorithms heuristics are applied that select the replacement strategy.

Besides the heuristic algorithms discussed above. Mauborgne [Mau98] applies approximation in the context of his work on Typed Decision Graphs (TDGs) which are a BDD variant. Mauborgne advocates widening TDGs for space, using an operator $\nabla(l, f)$ that takes, as input, a TDG that encodes a Boolean function $f$ and returns, as output, a TDG $g$ with at most $l$ nodes such that $f \models g$. The first widening he proposes is in $O(|G|^4)$ where $|G|$ is the number of nodes in the TDG. The cost of this algorithm is potentially prohibitive for TDGs possessing greater than $\approx 1000$ nodes, hence to improve efficiency, Mauborgne suggests a second widening that resembles those of Shiple and Ravi *et al.* This widening computes the TDGs $f_1, \ldots, f_n$ obtained by replacing each node $g$ with *true*. The $f_i$ are then filtered to remove those TDGs whose size exceed $|G|/2$. Of the remaining $f_i$, an $f_{max}$ is selected which "gives best results" [Mau98] and the widening is reapplied to $f_{max}$ if its TDG contains more than $l$ nodes.

An interesting development in the widening of Boolean formulae for ROBDD representation has been that of Schachte and Søndergaard [SS06]. Their work considers the problem of computing the strongest Boolean function in a given class that over-approximates a given Boolean function. The classes of function considered include those of *definite* [AMSS98] and *monotone* [Pos41] Boolean functions. The paper also addresses the decision problem of whether a given formula resides in a given class. The authors provide constructive answers, by means of several elegant ROBDD algorithms to the above problems for a number of classes of Boolean formulae. The algorithms they propose define functions which possess rich algebraic properties and actually correspond to *closure* operators on the domain of Boolean formulae. Although complexity theoretic issues still remain, these algorithms are potentially useful as widenings though no quantitive measurements are presented in the work.

### 5.4.2 Prime Implicants

The problem of computing the set of all prime implicants of a Boolean function was first mentioned by Shannon [Sha48]. However, the problem did not come to prominence until it was realised that prime implicants play a crucial role in computing the minimal DNF representation of a Boolean circuit – the problem addressed by the Quine McCluskey procedure [Qui52]. The complexity of the Quine McCluskey procedure was long believed

to be in $\mathrm{NP}^{\mathrm{NP}}$ since the problem was thought to be a two stage process. The first stage requiring the computation of all prime implicants, and the second stage computing a minimal cover over the implicants. The second stage is known to be NP-complete [Kar72] whereas until the result of Strzemecki [Str92], the generation of all prime implicants was assumed to be NP-complete. As Strzemecki showed, the complexity of computing all prime implicants of a Boolean function $f$ is in fact pseudo-polynomial in the number of truth assignments of the Boolean function [Str92]. The algorithm requires the generation of the truth table of the input Boolean function and although the algorithm is polynomial in the number of truth assignments, the number of truth assignments can in turn be exponential in the number of variables. Hence, the algorithm is not practical for large Boolean formulae.

Since ROBDDs are an efficient representation of Boolean formulae, the design of algorithms for efficient prime implicant generation for ROBDDs has attracted much interest. The number of prime implicants for a Boolean function over $n$ variables is bounded by $\frac{3^n}{\sqrt{n}}$ (and this bound is tight) [CM78] implying that any efficient prime enumeration algorithm has to address the problem of how to compactly represent the set of implicants. Coudert and Madre [CM92a] gave an elegant solution to this problem by representing the prime implicants themselves as an ROBDD. Then the complexity of prime enumeration is not necessarily reliant on the number of implicants themselves, but the number of nodes of the ROBDD required to express them. Alas, a detailed analysis of the complexity of their algorithm has not been forthcoming and it is unknown whether the algorithm is polynomial in the size of the input ROBDD [Cou] (although it is conjectured that it is not [Hay95]). Further afield algorithms have been proposed to compute the shortest prime implicant of a Boolean function using techniques such as *Integer Linear Programming* (ILP) [Piz96] and SAT-based branch-and-bound [MOS98].

Quite independent of the observation of using prime implicants of progressively larger length to iteratively widen ROBDDs, Lahiri *et al.* [LNO06] employ cubes of increasing length to compute over-approximations of logical formulae over propositional atoms drawn from a parameterised theory $T$. The authors show how to adapt a decision procedure based on the DPLL($T$) framework for SAT Modulo Theories (SMT) [NOT06] to compute abstractions over cubes of increasing size $k$, and demonstrate that this process is efficient for small $k$. This approach applies the same fundamental idea as widening with prime implicants, albeit working with arbitrary predicates rather than merely propositional variables.

## 5.5 The Widening

To decouple the specification of the widening from implementation concerns, we first specify how to widen Boolean functions for both space and time using prime implicants.

### 5.5.1 Widening for Space

As stated in the related work survey in §5.4, the ROBDD approximation algorithms of Shiple [Shi96] and Ravi *et al* [RMSS98] seek to improve the density of an ROBDD defined as the ratio of minterms in the represented function to the number of nodes in the representing ROBDD. Both algorithms identify the non-dense sub-ROBDDs within an ROBDD and substitute them with other sub-ROBDDs which are denser but possess more models. Ultimately this culminates in a dense over-approximation. Although this approach is well-intended, it has the following distinct disadvantages:

- density comparisons and ROBDD restructuring is limited to those sub-ROBDDs that actually arise in the ROBDD. The structure of the ROBDD to be approximated is heavily dependant upon the variable ordering employed (§2.2.1) hence differing approximations will be obtained for different variable orderings,
- although density is a natural measure of the efficiency of an ROBDD representation of a Boolean function, an approximation algorithm that endeavours to increase density alone may produce inaccurate approximations. For instance, suppose that a Boolean function over 8 variables has $2^7$ minterms and is represented by 16 nodes, thus its density is $\frac{2^7}{16} = 8$. The trivial approximation *true* can be represented as a single node, and therefore has density $\frac{2^8}{1} = 256$, yet this over-approximation conveys no information at all. Thus it is not prudent to base widening on density alone.

The observation is that prime implicants are natural variable order independent candidates for reasoning about density whilst retaining accuracy. To illustrate this, consider a Boolean function $f \in \mathbb{B}_X$ and the set of implicants $S = \{p \mid p \models f\}$ of $f$. Observe that any $S' \subseteq S$ is a sound under-approximation of $f$ in the sense that $\vee S' \models f$ where $\vee S' = \bigvee_{p \in S'} p$. However, different $S'$, even of the same size, can yield better approximations. For instance, consider an implicant $p \in S$ and a prime implicant $p'$ strictly contained within it, that is, $p \models p'$ and $p \neq p'$. Then $|p'| < |p|$. Hence $p'$ contributes $2^{n-|p'|}$ minterms to $f$ whereas $p$ contributes only $2^{n-|p|}$. Thus $p'$ is a better candidate for inclusion in $S'$ than $p$. Moreover, since $p'$ is shorter than $p$, it is likely to contribute a shorter path in an ROBDD that represents $\vee S'$. The following family of widening operators draw together these ideas to compute a sound over-approximation by combining negation with systematic under-approximation.

**Definition 5.1.** *The family of operators* $\nabla_k : \mathbb{B}_X \to \mathbb{B}_X$ *where* $k \in \mathbb{N} \cup \{0\}$ *are defined by* $\nabla_k(f) = \bigwedge\{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\}$

We now proceed to prove the properties of the widening $\nabla_k$. The proposition asserts that $\nabla_k$ is anti-monotonic in its parameter $k$ and hence $\nabla_k$ is uniformly more precise than $\nabla_{k-1}$. Furthermore, in the limit, $\nabla_k(f)$ converges onto $f$ from above. We show

that the widening is also monotonic in its argument $f$, that is, for $f, f' \in \mathbb{B}_X$ such that $f \models f'$ then $\nabla_k(f) \models \nabla_k(f')$.

**Proposition 5.1.** *Suppose* $|X| = n$. *Then*

- *Given* $f \in \mathbb{B}_X$ *then* $f = \nabla_n(f) \models \nabla_{n-1}(f) \models \ldots \models \nabla_0(f) = \mathit{true}$.
- *Given* $f, f' \in \mathbb{B}_X$ *such that* $f \models f'$ *and* $0 \leq k \leq n$ *then* $\nabla_k(f) \models \nabla_k(f')$.
- *Given* $f, f' \in \mathbb{B}_X$ *such that* $f \models f'$ *and* $0 \leq \ell < k \leq n$ *then* $\nabla_k(f) \models \nabla_\ell(f')$.

*Proof.*

- It is well-known that $f = \bigvee \mathit{primes}(f)$ [CH09], hence in the limit $f = \nabla_n(f)$. To prove for $0 \leq k < n$. Observe,

  $$\{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\} \subseteq \{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k+1\}$$

  hence it follows that,

  $$\bigwedge\{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k+1\} \models \bigwedge\{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\}$$

  and thus $\nabla_{k+1}(f) \models \nabla_k(f)$. Finally observe $\nabla_0(f) = \wedge\emptyset = \mathit{true}$ as required.

- Let $0 \leq k \leq n$, $p' \in \mathit{primes}(\neg f')$ and $|p'| \leq k$. Since $p' \models \neg f'$ by definition, further by assumption we have $f \models f'$ then $\neg f' \models \neg f$ hence $p' \models \neg f' \models \neg f$. Furthermore, there exists some $p \in \mathit{primes}(\neg f)$ such that $p' \models p$ thus $|p| \leq |p'| \leq k$. Since $p' \models p$, $\neg p \models \neg p'$, hence,

  $$\nabla_k(f) = \bigwedge\{\neg p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\} \models \neg p'$$

  Therefore $\nabla_k(f) \models \nabla_k(f')$ as required.

- Holds trivially by combining the above two cases. □

From the above proposition we now proceed to show that the widening $\nabla_k$ constitutes an *upper closure* operator on the complete lattice of Boolean formulae, and therefore the widening may be used as the basis for defining a Galois connection [CC79]. This is ironic since widening is often deployed when a Galois connection does not exist.

**Definition 5.2** (Upper Closure Operator)**.** *An upper closure operator (uco) is a function* $\rho : L \to L$ *on a complete lattice* $\langle L, \sqsubseteq, \bot, \top, \sqcap, \sqcup\rangle$ *such that* $\rho$ *is,*

- *monotonic: if* $x \sqsubseteq y$ *then* $\rho(x) \sqsubseteq \rho(y)$ *for all* $x, y \in L$ *and,*
- *extensive:* $x \sqsubseteq \rho(x)$ *for all* $x \in L$ *and,*

- *idempotent:* $\rho(x) = \rho(\rho(x))$ *for all* $x \in L$.

**Corollary 5.1.** *The widening* $\nabla_k$ *is an upper closure operator on the complete lattice* $\langle \mathbb{B}_X, \models, 0, 1, \vee, \wedge, \neg \rangle$.

*Proof.* The first two requirements follow as a consequence of Proposition 5.1. Suppose $f' = \nabla_k(f)$ for some $f \in \mathbb{B}_X$ and $k \in \mathbb{N} \cup \{0\}$. Furthermore, let $f'' = \nabla_k(f')$ then $f \models f'$ and $f' \models f''$ thus $\neg f'' \models \neg f' \models \neg f$. Observe, by De Morgans,

$$\neg f' = \vee\{p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\}$$
$$\neg f'' = \vee\{p \mid p \in \mathit{primes}(\neg f') \wedge |p| \leq k\}$$

since $f' \models f''$ it is sufficient to show $f'' \models f'$, or equivalently, $\neg f' \models \neg f''$. By Blake canonical form [CH09], this sufficient condition can be reduced to the requirement $\mathit{primes}(\neg f') \subseteq \mathit{primes}(\neg f'')$. Let $p \in \mathit{primes}(\neg f')$. Since $\neg f' = \vee\{p \mid p \in \mathit{primes}(\neg f) \wedge |p| \leq k\}$ it follows that $|p| \leq k$. Since $\neg f'' = \vee\{p \mid p \in \mathit{primes}(\neg f') \wedge |p| \leq k\}$ it follows that $p \models \neg f''$. Suppose for the sake of a contradiction that $p \notin \mathit{primes}(\neg f'')$. Thus there exists some $p' \models \neg f''$ such that $p \models p'$ and $p \neq p'$. Hence $|p'| \leq k$, and $p' \models \neg f'$ since $\neg f'' \models \neg f'$. Thus $p \notin \mathit{primes}(\neg f')$ and $p \in \mathit{primes}(\neg f'')$ as required. □

### 5.5.2 Widening for Time

To demonstrate the rôle of prime implications in widening for time, consider a chain of Boolean functions $\{f_1, f_2, \ldots\} \subseteq \mathbb{B}_X$ such that $f_{i+1} = F(f_i)$ for some $F : \mathbb{B}_X \to \mathbb{B}_X$ where $F$ is a monotonic operator. The problem is to extract an invariant from that chain, that is, find a function $g$ such that $f_i \models g$ for all $f_i$, hence $g$ is a sound over-approximation of all $f_i$. Such an invariant can be found, whilst applying $F$ only a bounded number of times, by constructing a set of $m$ Boolean functions $S_1 = \{g_1, \ldots, g_m\}$ such that $f_1 \models g_i$ for all $g_i$. This set is then iteratively pruned until stability is reached. This is realised by constructing the chain of sets $S_{i+1} = \{g \in S_i \mid F(\wedge S_i) \models g\}$. By construction $f_1 \models \wedge S_1$ since $f_1 \models g$ for all $g \in S_1$. Moreover, since $F$ is monotonic, it follows that $f_{i+1} = F(f_i) \models F(\wedge S_i) \models \wedge S_{i+1}$. Hence, for all $i \geq 1$, $f_i \models \wedge S_i$.

If $S_l$ denotes the limit, that is $S_l = S_{l+1}$, then $f_i \models \wedge S_l$ for all $f_i$, hence $\wedge S_l$ is an invariant. The key point about this construction is that $F$ is applied at most $m$ iterations rather than possibly $2^{|X|}$ times. This gives a performance guarantee and a parameter $m$ that can be increased (if necessary) to improve precision. This merely leaves the problem of constructing the initial set $S_1$.

An uninformed approach to computing $S_1$ is to extract $m$ arbitrary implicants of $\neg f_1$, that is, $p \models \neg f_1$. Then each $\neg p$ is a clause of $f_1$. However, consider a prime implicant $p'$ of $\neg f_1$ such that $p \models p'$. Then $\neg p' \models \neg p$, therefore substituting a prime implicant $p'$ for $p$ we obtain a more accurate initial $\wedge S_1$, without increasing its size. This motivates

constructing $S_1$ from prime implicants. Furthermore, consider two prime implicants $p$ and $p'$ such that $|p'| < |p|$. Then $p'$ is a more propitious candidate for inclusion in $S_1$ since the clause $\neg p'$ possesses fewer minterms than $\neg p$ which motivates a greedy approach to constructing $S_1$ in terms of prime implicants of minimal length.

One may wonder whether a bound $k$ on the length of prime implicants, induces a bound on the number $m$ of primes, and hence a bound on the number of iterates. A straightforward relationship between $k$ and $m$ follows from the observation that there are $\binom{n}{1}2^1, \binom{n}{2}2^2, \ldots, \binom{n}{k}2^k$ different cubes of length $1, 2, \ldots, k$ respectively where $\binom{i}{j}$ denotes the binomial coefficient $\binom{i}{j} = \frac{i!}{(i-j)!j!}$. Hence a bound on $m$ is $\min(\{2^n, \sum_{i=1}^{k}\binom{n}{k}2^k\})$ where $n = |X|$. The $2^n$ component of the bound follows from the observation that there are $2^n$ possible truth assignments hence the maximum number of times a Boolean formula can be weakened is $2^n$. However, by adapting an argument relating to anti-chains of implicants [CM78, proof of Theorem 2.2], the following tighter bound can be obtained on the number of primes whose length does not exceed $k$:

**Proposition 5.2.** $|\{p \in \mathit{primes}(f) \mid |p| \leq k\}| \leq \max(\{\binom{n}{1}2^1, \ldots, \binom{n}{k}2^k\})$

*Proof.* Let $f \in \mathbb{B}_X$, $X = \{x_1, \ldots, x_n\}$, the set $C$ denote the set of cubes over $X$ and $P = \{p \in \mathit{primes}(f) \mid |p| \leq k\}$. $P$ is anti-chain of $\mathbb{B}_X$ and also $C$. It has been shown [KEL71] that in a poset such as $\langle C, \models\rangle$, there exists a maximal anti-chain which is invariant under any isomorphism of $C$. Let $A$ be such an anti-chain. Now let $c, c' \in A$ such that $|c| = |c'|$ and consider a mapping $F : Y \to Y$ where $Y = \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$ such that $F(x_i) = \neg F(\neg x_i)$. Suppose that $F(c) = c'$ where $F$ is extended from $Y$ to $C$ in the natural way. Since $F$ is an automorphism and $c' \in A$, it follows that $\{c'' \in C \mid |c| = |c''|\} \subseteq A$. Since $A$ is an anti-chain, then if $p \in C$ and $|p| < |c|$ then there exists $c'' \in A$ such that $p \models c''$ and $p \neq c''$. Hence $p \notin A$. Similarly, if $|p| > |c|$ then $p \notin A$. Therefore $|A| = \binom{n}{|c|}2^{|c|}$ and, since $|P| \leq |A|$, the result follows. □

Whenever $3k \leq 2(n+1)$ the above bound on $m$ collapses to $\binom{n}{k}2^k$. This follows since,

$$\binom{n}{1}2^1 \leq \ldots \leq \binom{n}{k-1}2^{k-1} \leq \binom{n}{k}2^k \iff \frac{1}{(n-k+1)} \leq \frac{2}{k} \iff 3k \leq 2(n+1)$$

Observe that the complexity bound in Proposition 5.2 implies that the maximum chain length permitted is fixed parameter tractable (FPT [FG06]) with respect to $k$ since the chain length becomes polynomial when $k$ is fixed. However, because this bound is so conservative, a more pragmatic tactic is needed for generating the shortest $m$ prime implicants. One such tactic is to compute all prime implicants of length 1 for $f_1$, then all primes whose length does not exceed 2, then all primes whose length does not exceed 3 *etc*, until $m$ prime implicants are discovered or some computational resource bound is exceeded.

In the following section we present a new algorithm for solving this specific form of the prime implicant enumeration problem.

## 5.6 Implementation of the Widening

In this section we present an algorithm with which to implement the widening, the algorithm appears to be practical in the general case despite complexity theoretic questions remaining unanswered.

### 5.6.1 The Algorithm

In [CM92a], Coudert and Madre gave an elegant algorithm for computing all the prime implicants of a Boolean function presented as an ROBDD. The advantage of their approach is that the primes themselves are, in turn, represented in an ROBDD. Hence, the complexity of prime implicant enumeration is not necessarily reliant on the number of implicants (which can grow to $\frac{3^n}{\sqrt{n}}$ for a function over $n$ variables) but the size of the intermediary ROBDDs.

#### The Meta-Product Representation

To represent the prime implicants uniquely whilst canonicity is retained, an encoding known as the Meta-Product is used. The Meta-Product representation was first defined in [CM92a], the purpose of which is to have a canonical representation of sets of cubes that can be efficiently represented and manipulated using ROBDDs. Hence as noted in [CM93b], theoretically, $\lceil \log_2 3^n \rceil$ Boolean variables are sufficient to represent any prime implicant thus an ROBDD over $\lceil \log_2 3^n \rceil$ variables is sufficient to represent any set of prime implicants. The Meta-Product representation defines a simple encoding over $2n$ variables. The first $n$ variables, denoted $O = \{o_1, \ldots, o_n\}$, the occurrence variables; the second $n$, denoted $S = \{s_1, \ldots, s_n\}$, the sign variables. Formally, we encode a prime implicant $p$ by $\sigma(p) = \bigwedge\{o_i \wedge s_i \mid p \models x_i\} \wedge \bigwedge\{o_i \wedge \neg s_i \mid p \models \neg x_i\}$. This mapping lifts to a Boolean function by $\sigma(f) = \bigvee\{\sigma(p) \mid p \in \mathit{primes}(f)\}$.

**Example 5.1.** *Consider* $f = x_1 \rightarrow x_2$ *then* $\sigma(f) = \bigvee\{\sigma(p) \mid p \in \mathit{primes}(x_1 \rightarrow x_2)\} = \bigvee\{\sigma(\neg x_1), \sigma(x_1 \wedge x_2)\} = (o_1 \wedge \neg s_1) \vee (o_1 \wedge s_1 \wedge o_2 \wedge s_2)$.

#### Prime Implicant Computation

The essence of the Coudert and Madre [CM92a,CM92b,CM93b] scheme is a transformation mapping an ROBDD representing $f$ over the variables $X$ to another representing a function $f'$ in the Meta-Product representation. Coudert and Madre's algorithm recursively builds $f'$ from $f$ utilising the observation that the prime implicants of a Boolean

function $f$ where,

$$f = (\neg x_i \wedge f|_{x_i \leftarrow 0}) \vee (x_i \wedge f|_{x_i \leftarrow 1})$$

where $x_i = index(f)$ can be constructed from the combination of the primes of the functions $f|_{x_i \leftarrow 0}$ and $f|_{x_i \leftarrow 1}$. This process gives a way to compute the primes of $f$ without explicitly computing all its implicants [CM92a]. Algorithm 9 gives the pseudo-code of Coudert and Madre's algorithm [CM92a]. This algorithm computes the Meta-Product $\sigma(f)$ for any given Boolean function $f$.

**Algorithm 9** `PRIMES`$(f)$

**Require:** $f \in ROBDD_X$

1: $x_i \leftarrow$ `index(f)`
2: $g \leftarrow$ `PRIMES`$(f|_{x_i \leftarrow 0} \wedge f|_{x_i \leftarrow 1})$
3: $g' \leftarrow$ `PRIMES`$(f|_{x_i \leftarrow 1}) \wedge \neg g$
4: $g'' \leftarrow$ `PRIMES`$(f|_{x_i \leftarrow 0}) \wedge \neg g$
5: **return** $((\neg o_i \wedge g) \vee (o_i \wedge s_i \wedge g') \vee (o_i \wedge \neg s_i \wedge g''))$

### Refined Prime Implicant Computation

In this section we propose a refinement to the algorithm of Coudert and Madre [CM92a, CM92b, CM93b]. The refinement specialises Coudert and Madre's algorithm to enumerate only those prime implicants whose length does not exceed some predefined constant $k \in \mathbb{N} \cup \{0\}$ such that $k \leq n$ where $n = |X|$. The new insight is that it is possible to build $f'$ from $f$ whilst enforcing the cardinality constraint $\sum_{i=1}^{n} o_i \leq k$ without explicitly enumerating the implicants of $f$. Hence the complexity of the refined algorithm is not necessarily reliant on the number of implicants of length less or equal to $k$, but as before, the size of the intermediary ROBDDs. Moreover, it is hoped that by removing primes of excessive length on each recursive step, the size of intermediary ROBDDs will be significantly reduced thus providing a speed up (since ROBDD operations are predominantly quadratic in the size of their inputs). The following algorithm builds toward the refined algorithm by generating an ROBDD which expresses the cardinality constraint. The constraint is realised as a cascade of $n$ full-adders that together output the sum expressed in $\lceil \log_2 n \rceil$ bits. The bits are then constrained so as to not exceed $k$.

In Algorithm 10, the bound $k$ is represented as an array of $\lceil \log_2 n \rceil$ bits $k[i]$, that is the binary representation of $k$. The first loop (lines $\#1 - \#3$) initialises the elements of the temporary array $sum[i]$ to $false$. The second loop (lines $\#4 - \#11$) iteratively calculates $o_1 + \ldots + o_n$ and stores the result in the temporary array $sum$. The $i^{th}$ iteration of the loop initialises the carry $c$ to be $o_i$ and then proceeds to add the carry into the sum that has accumulated thus far. The formula $sum[j] \oplus c$ merely denotes the exclusive-or of the $j^{th}$ bit of $sum$ with the carry $c$. The third loop (lines $\#13 - \#15$) constrains the array $sum$ to not exceed the $k$ vector. Algorithm 11 details how this

**Algorithm 10** CONSTRAIN$(k)$

**Require:** $k \in ROBDD^{\log_2 n}$
1: **for** $i \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
2: $sum[i] \leftarrow false$
3: **end for**
4: **for** $i \leftarrow 1$ **to** $n$ **do**
5: $c \leftarrow o_i$
6: **for** $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
7: $c' \leftarrow c \wedge sum[j]$
8: $sum[j] \leftarrow sum[j] \oplus c$
9: $c \leftarrow c'$
10: **end for**
11: **end for**
12: $f \leftarrow false$
13: **for** $i \leftarrow 1$ **to** $n$ **do**
14: $f \leftarrow (\neg sum[i] \wedge k[i]) \vee ((sum[i] \leftrightarrow k[i]) \wedge f)$
15: **end for**
16: **return** $f$

**Algorithm 11** PRIMESLEQ$(f, k)$

**Require:** $f \in ROBDD_X$, $k \in \mathbb{B}^{\log_2 n}$
1: $x_i \leftarrow$ index(f)
2: $g \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 0} \wedge f|_{x_i \leftarrow 1}, k)$
3: $g' \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 1}, k) \wedge \neg g$
4: $g'' \leftarrow$PRIMESLEQ$(f|_{x_i \leftarrow 0}, k) \wedge \neg g$
5: **return** $((\neg o_i \wedge g) \vee (o_i \wedge s_i \wedge g') \vee (o_i \wedge \neg s_i \wedge g'')) \wedge$ CONSTRAIN$(k)$

constraint can be integrated in the algorithm of Coudert and Madre [CM92a].

Computing CONSTRAIN$(k)$ is not prohibitively expensive, in fact, Bartzis and Bultan [BB03] show that the constraint $\sum_{i=1}^{n} o_i \leq k$ can be constructed in $O(n^2)$ time and space. Recently, this bound was reduced to $O(k(n-k))$ [HLS05].

Algorithm 11 repeatedly imposes the cardinality constraint on each recursive step, which is hoped will trim the size of all intermediate ROBDDs. The astute reader will notice that each call to PRIMESLEQ operates on a sub-ROBDD that is only defined over the variable set $\{x_j, \ldots, x_n\}$ for $i < j$. However, CONSTRAIN$(k)$ imposes a constraint over the variable set $\{x_1, \ldots, x_n\}$. This is no error since $\sum_{i=1}^{n} o_i \leq k$ entails $\sum_{i=j}^{n} o_i \leq k$ and therefore it is not necessary to manufacture a different cardinality constraint for each level in the ROBDD.

When widening for time, it is necessary to extract $m$ prime implicants from the transformed ROBDD $f'$. This can be accomplished by a partial, depth-first traversal that sweeps the ROBDD until $m$ prime implicants of minimal length have been retrieved. When widening for space, an ROBDD representation of the over-approximation

is required. The following algorithm details how this can be constructed by applying existential quantification:

**Algorithm 12** PRIMES2BDD($f$)

**Require:** $f \in ROBDD_X$
1: **for** $i \leftarrow 0$ **to** $n$ **do**
2: $\quad f' \leftarrow \exists_{s_i}(\exists_{o_i}(f \wedge (o_i \rightarrow (x_i \leftrightarrow s_i))))$
3: $\quad f \leftarrow f'$
4: **end for**
5: **return** $f$

### 5.6.2 Complexity Issues

As we stated in §5.4.2, a detailed analysis of the complexity of Coudert and Madre's algorithm has not been forthcoming and it is unknown whether the algorithm is polynomial in the size of the input ROBDD [Cou]. Instead of providing an argument that the algorithm is polynomial we provide a reduction which suggests that both Coudert and Madre's algorithm and the algorithmic refinement proposed in this chapter are unlikely to be polynomial. The reduction asserts that the problem of finding the shortest implicant of a Boolean function presented as an ROBDD is at least as hard as a problem pertaining to variable ordering. Specifically, we formulate the SHORTEST IMPLICANT problem and show this to be reducible to the VARIABLE ORDER SHORTEST PATH problem.

**Definition 5.3.** SHORTEST IMPLICANT[ROBDD]
**Instance:** Given an ROBDD $f$ and an integer $k$.
**Problem:** Is there an implicant of $f$ that contains $k$ or fewer literals?

Notice that this problem is formulated in terms of the existence of a shortest implicant rather than a shortest prime implicant. It is sufficient to focus on this simplified problem because if there exists an implicant of size at most $k$ then there immediately exists a prime implicant of length at most $k$ contained within it.

Consider the following problem:

**Definition 5.4.** VARIABLE ORDER SHORTEST PATH
**Instance:** Given an ROBDD $f$ over the set of variables $X = \{x_1, x_2, \ldots, x_n\}$ and an integer $k$.
**Problem:** Is there a reordering of the elements of $X$ such that the resultant ROBDD $f'$ has a path to *true* of length at most $k$?

**Proposition 5.3.** VARIABLE ORDER SHORTEST PATH $\leq_T$ SHORTEST IMPLICANT[ROBDD]

*Proof.* Given an instance of the SHORTEST IMPLICANT[ROBDD] problem. Suppose there exists an implicant $p$ such that $|p| \leq k$. Let $\pi : X \to \mathbb{N} \cup \{0\}$ be any injective function such that $\pi(x_i) < \pi(x_j)$ for all $x_i \in var(p) \wedge x_j \in X \setminus var(p)$. Reordering $f$ w.r.t. order $\pi$ yields an ROBDD $f'$ which contains a path to *true* of length at most $k$.

Therefore VARIABLE ORDER SHORTEST PATH $\leq_T$ SHORTEST IMPLICANT[ROBDD] as required. □

This reduction suggests that the complexity of PRIMES($f$) is not polynomial. To see this, observe that if $f'$ can be computed in polynomial time, then it is possible to compute the shortest prime implicant in polynomial time. This follows because it is possible to identify the shortest prime implicant in polynomial time as demonstrated by the following function.

$$occ(f) = \begin{cases} occ(f|_{o_j \leftarrow 1}) + 1, & \text{if } index(f) = o_j \wedge f|_{o_j \leftarrow 0} = false \\ occ(f|_{o_j \leftarrow 0}), & \text{else if } index(f) = o_j \wedge f|_{o_j \leftarrow 1} = false \\ \min(occ(f|_{o_j \leftarrow 0}), occ(f|_{o_j \leftarrow 1}) + 1), & \text{else if } index(f) = o_j \\ \min(occ(f|_{s_j \leftarrow 0}), occ(f|_{s_j \leftarrow 1})), & \text{else if } index(f) = s_j \\ 0, & \text{otherwise.} \end{cases}$$

The function annotates each node in $f'$ with a count of the minimum number of positive $o_i$ variables occurring along any path emanating from the node. At the root of $f'$, this count corresponds to the length of the shortest prime implicant of the function $f$. The shortest implicant can then be found in $O(n)$ time by traversing $f'$ from the root, selecting the child whose root is labelled with the smallest count.

## 5.7 Experimental Results

To assess the precision of the widening along with the practicality of the proposed implementation, we implemented the algorithms of §5.6.1 within CUDD [Som05]. The package contains implementations of both the algorithms of Shiple [Shi96] and Ravi *et al.* [RMSS98] which, following the CUDD naming scheme, will henceforth be referred to as `bddOverApprox` and `remapOverApprox` respectively. Table 11 presents details of the Boolean functions, drawn from the MCNC and ISCAS benchmark circuits, used to assess the widening. For ease of reference, all Boolean functions are labelled with a numeric identifier. The second and third columns give the circuit name and specific output number taken from the circuits; outputs were selected so as to evaluate the widening on ROBDDs with varying size. The fourth, fifth, sixth and seventh columns respectively give the number of variables, number of ROBDD nodes, the number of truth assignments of the Boolean function represented by the ROBDD and the density of the ROBDD. All experiments were performed on an UltraSPARC IIIi 900MHz based

| ID | Circuit | # | \|X\| | size | minterms | density |
|---|---|---|---|---|---|---|
| 1. | pair | 177 | 51 | 26253 | $1.86 \times 10^{14}$ | $7.08 \times 10^{9}$ |
| 2. | | 182 | 53 | 33190 | $8.12 \times 10^{14}$ | $2.45 \times 10^{10}$ |
| 3. | mm9b | 420 | 31 | 94328 | $1.61 \times 10^{9}$ | $1.71 \times 10^{4}$ |
| 4. | | 421 | 31 | 96875 | $1.62 \times 10^{9}$ | $1.67 \times 10^{4}$ |
| 5. | s9234 | 288 | 76 | 655192 | $3.59 \times 10^{22}$ | $5.48 \times 10^{16}$ |
| 6. | | 488 | 75 | 1304371 | $1.95 \times 10^{22}$ | $1.49 \times 10^{16}$ |
| 7. | rot | 149 | 53 | 1315 | $5.18 \times 10^{15}$ | $3.94 \times 10^{12}$ |
| 8. | | 172 | 55 | 1700 | $1.08 \times 10^{16}$ | $6.35 \times 10^{12}$ |

Table 11: Benchmark formulae

system, equipped with 16GB RAM, running the Solaris 9 Operating System, and using `getrusage` to calibrate CPU usage in seconds.

### 5.7.1 Our Method

Figure 8 presents the time required to apply Algorithm 11 and then Algorithm 12 to the benchmarks in Tables 11 for various $k$. (Note that this time is dominated by the cost of applying Algorithm 11 and therefore the times reported in the table closely tally with the times required to apply Algorithm 11 and then walk the ROBDD to extract a bounded number of primes).

Interestingly, Coudert and Madre [CM92a] suggest that "[their] procedures have costs that are independent of the sizes of [the prime] sets", "since there is no relation between the size of a set and the size of the [ROBDD] that denotes it". However, this does not square with the results which suggest that the size of the ROBDDs depends, at least to some extent, on the number of primes that it represents providing empirical evidence as to the complexity of the problem at hand. This is witnessed by the sharp increase in run time that occurs for some circuits as $k$ increases. However, the crucial point to observe is not that the run time spikes, but the degree of precision achieved before the escalation in complexity. To this end, Figure 9 plots the ratio of truth assignments of the original Boolean function against that of the approximation for increasing values of $k$. Observe that the quality of the approximation rapidly converges onto *true* as $k$ increases, hence the approximation exhibits the diminishing property that is considered to be desirable in an anytime algorithm. The diminishing property suggests the tactic of incrementally increasing $k$ until either the precision is acceptable or a timeout is reached. Applying this tactic achieves precision rates of $70, 80$, and $90\%$ yielding run times of less than $5, 20$ and $60$ seconds respectively. On the other hand, repeatedly incrementing $k$ until the accumulated run time exceeds 30 seconds, achieves precision rates for benchmarks $1-8$ of $99, 99, 99, 99, 99, 92, 96, 95\%$ respectively.

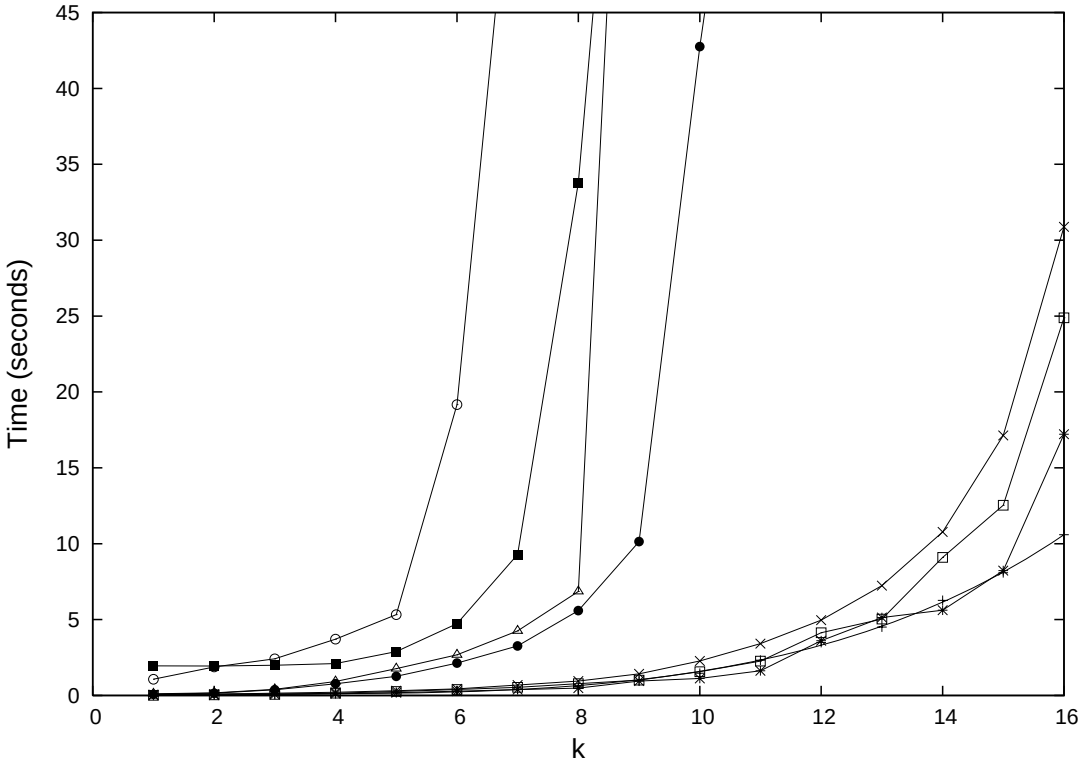


Figure 8: Time against $k$

This realises an anytime approach to prime generation and ROBDD approximation in which the quality of the result is limited only by the quantity of resource available. Incrementing $k$ until at least 1024 prime implicants are found (which if anything is rather high for the purposes of program analysis), requires the following values of $k$ : $5, 5, 7, 7, 5, 6, 7, 7$ for the 8 benchmark circuits. It should be noted that these figures are, if anything, rather pessimistic for many types of program analysis. For example, in the context of groundness analysis that is widely used in logic programming, it has been observed that the vast majority of clauses that arise during analysis are very small in length [HACK00]. This implies that widening with small $k$ is unlikely to have any discernible impact on the overall precision.

The value of an approximation algorithm has traditionally been reported in terms of density [Shi96, RMSS98] which gives an indication as to the compactness of the approximating ROBDD. Figure 10 thus reports how the density varies with $k$. By comparing the densities reported in Table 11 against those presented in the graph, it can be seen that the widening can significantly improve on the density of the original ROBDD resulting in a much more compact approximation.

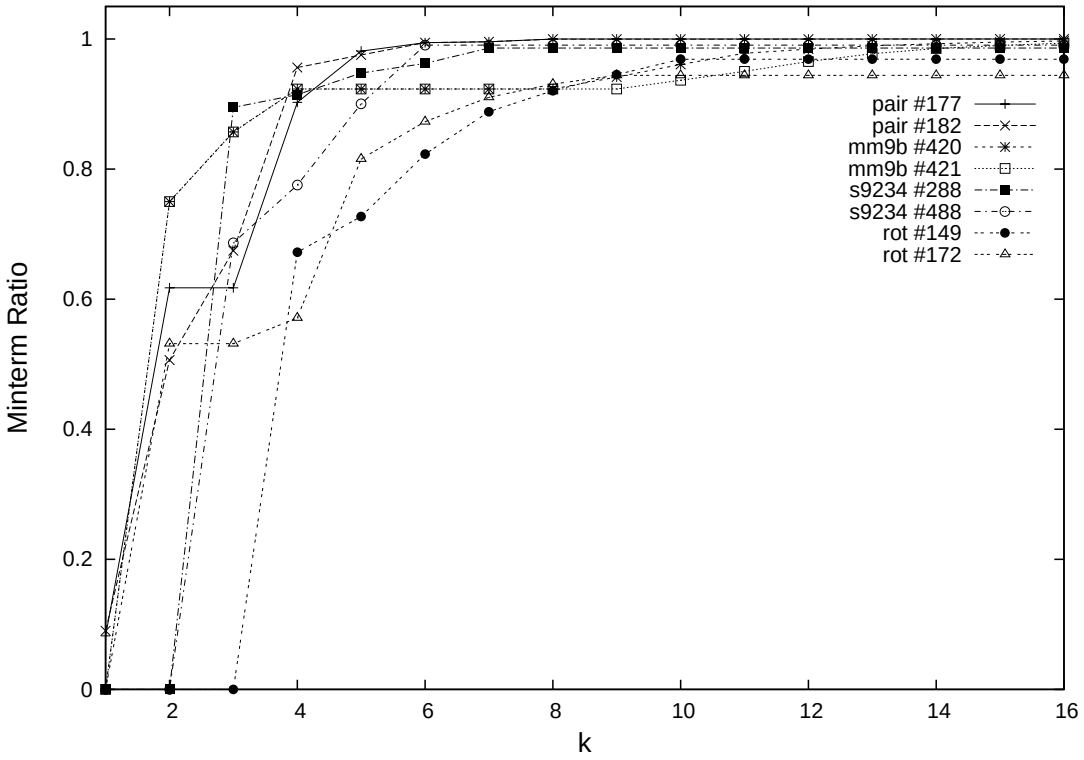


Figure 9: Minterm ratio against $k$

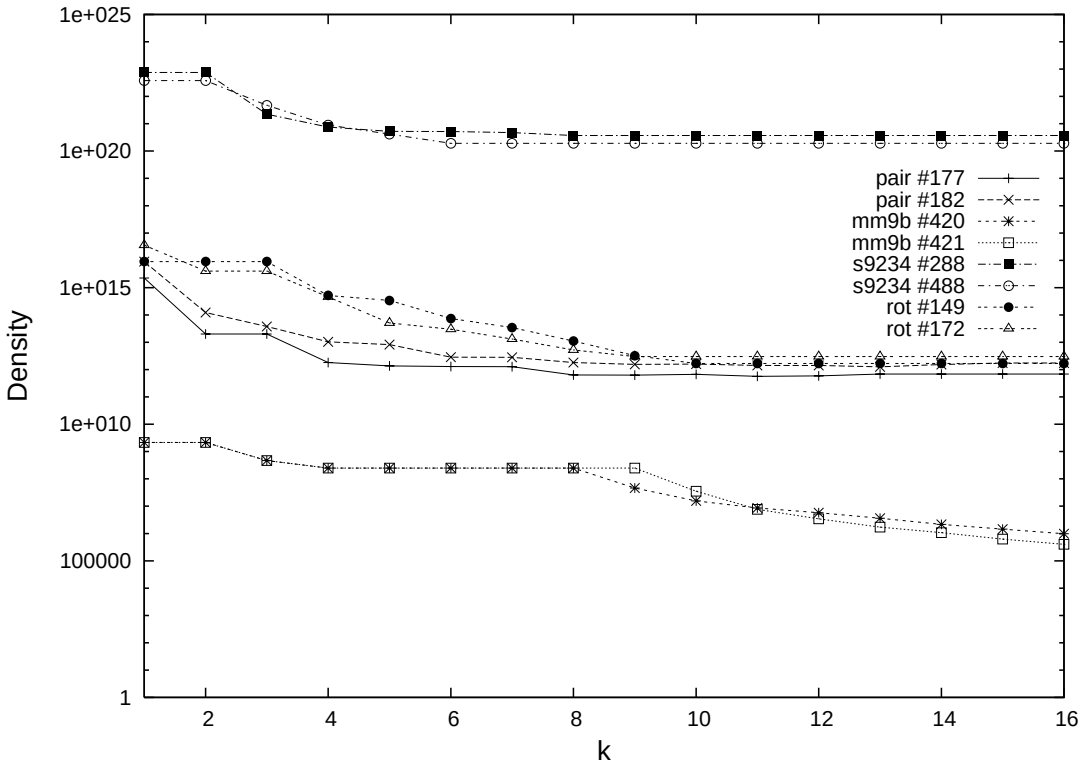


Figure 10: Density against $k$

### 5.7.2 Comparison Against Existing Methods

Table 12 summaries the results obtained by the `bddOverApprox` and `remapOverApprox` algorithms on the circuits in the benchmark suite. The table is partitioned horizontally, into three groups of rows according to whether the `bddOverApprox` algorithm, `remapOverApprox` algorithm, or the widening algorithm proposed in this work was applied. The second and third columns give the size of the approximating ROBDD and the number of minterms in its underlying Boolean function. The fourth and fifth columns detail the ratio of these values with respect to the size and number of minterms in the original ROBDD (as given in Table 11). The `bddOverApprox` and `remapOverApprox` algorithms are parameterised by a quality parameter $q \in [0, 1]$, that specifies the minimal acceptable density improvement. That is, these algorithms ensure that the new density $d'$ satisfies $q \geq d/d'$ where $d$ is the density of the original ROBDD. As Shiple himself says [Shi96], "The `bddUnderApprox` method is highly sensitive to the [quality] parameter". Added to this, there is no clear way to choose $q$ so as to obtain a desired reduction in ROBDD size.

For purposes of comparison, we chose to reduce the size of an ROBDD by at least 50%, but ideally not significantly more than 50% (it was the desire to solve this particular analysis problem that motivated this study). Both `bddOverApprox` and `remapOverApprox` were called repeatedly under the bisection algorithm to search for a quality value that yielded an acceptable reduction in size. The algorithm terminated when the difference between the high and lower quality bounds was less than 0.01. The notes column gives the particular quality values that achieved the best ROBDD approximation and the time column presents the total time required to call bisection which, of course, was dominated by the time to approximate the ROBDDs. Despite the systematic use of bisection, the reduction in ROBDD size was often significantly more than 50%. This was due to the ROBDD collapsing at certain quality thresholds.

The lower rows of the Table 12 summarise the results of incrementing $k$ until a space reduction of at least 50% was obtained. The notes column gives the required values of $k$ and the cumulative execution time. Observe that the minterm ratios thus obtained compare favourably with those derived using `bddOverApprox` and `remapOverApprox` whilst the overall execution time is also reduced. Note that other variable orderings may give different results for the `bddOverApprox` and `remapOverApprox` whereas the approximation derived from our widening is constant for all variable orderings. As a sanity check, the widening was tested to verify that it delivered the same approximations under different variable orderings.

| ID | | Approximation size | Approximation minterms | Ratios size | Ratios minterms | Time | Notes |
|---|---|---|---|---|---|---|---|
| [RMSS98] | 1. | 8382 | $3.40 \times 10^{14}$ | 0.32 | 1.83 | 4.61 | **q:** 0.94 |
| | 2. | 9711 | $1.47 \times 10^{15}$ | 0.29 | 1.81 | 6.32 | **q:** 0.84 |
| | 3. | 933 | $1.88 \times 10^{9}$ | 0.01 | 1.16 | 10.85 | **q:** 0.75 |
| | 4. | 722 | $1.88 \times 10^{9}$ | 0.01 | 1.16 | 11.96 | **q:** 0.84 |
| | 5. | 15 | $5.68 \times 10^{22}$ | 0.01 | 1.58 | 1086.12 | **q:** 0.88 |
| | 6. | 11 | $2.89 \times 10^{22}$ | 0.01 | 1.49 | 2321.68 | **q:** 0.92 |
| | 7. | 91 | $7.30 \times 10^{15}$ | 0.07 | 1.41 | 1.13 | **q:** 0.96 |
| | 8. | 838 | $2.96 \times 10^{16}$ | 0.49 | 2.75 | 1.50 | **q:** 0.98 |
| [Shi96] | 1. | 8385 | $1.72 \times 10^{15}$ | 0.32 | 10.85 | 4.86 | **q:** 0.92 |
| | 2. | 9714 | $8.06 \times 10^{15}$ | 0.29 | 9.93 | 6.35 | **q:** 0.81 |
| | 3. | 933 | $1.88 \times 10^{9}$ | 0.01 | 1.16 | 12.39 | **q:** 0.75 |
| | 4. | 722 | $1.88 \times 10^{9}$ | 0.01 | 1.16 | 13.10 | **q:** 0.84 |
| | 5. | 15 | $5.68 \times 10^{22}$ | 0.01 | 1.58 | 1057.62 | **q:** 0.87 |
| | 6. | 11 | $2.89 \times 10^{22}$ | 0.01 | 1.49 | 2562.30 | **q:** 0.92 |
| | 7. | 168 | $8.10 \times 10^{15}$ | 0.13 | 1.56 | 1.25 | **q:** 0.92 |
| | 8. | 837 | $1.73 \times 10^{16}$ | 0.49 | 1.60 | 1.67 | **q:** 0.94 |
| §5.6 | 1. | 11027 | $2.06 \times 10^{14}$ | 0.42 | 1.11 | 0.58 | **k:** 5 |
| | 2. | 7301 | $8.32 \times 10^{14}$ | 0.22 | 1.03 | 0.85 | **k:** 6 |
| | 3. | 44334 | $1.68 \times 10^{9}$ | 0.47 | 1.02 | 6.38 | **k:** 12 |
| | 4. | 39718 | $1.69 \times 10^{9}$ | 0.41 | 1.05 | 8.19 | **k:** 11 |
| | 5. | 75 | $3.64 \times 10^{22}$ | 0.01 | 1.01 | 20.36 | **k:** 7 |
| | 6. | 103 | $1.96 \times 10^{22}$ | 0.01 | 1.01 | 47.53 | **k:** 6 |
| | 7. | 289 | $6.29 \times 10^{15}$ | 0.22 | 1.21 | 0.88 | **k:** 7 |
| | 8. | 527 | $1.09 \times 10^{16}$ | 0.31 | 1.01 | 1.66 | **k:** 7 |

Table 12: Comparison of approximation

## 5.8 Provably Polynomial Widening

The widening presented in §5.5 relies upon the generation of prime implicants. This problem was first addressed by Quine [Qui52] and, since then, there has been much interest in developing efficient prime implicant enumeration algorithms (interested readers should consult [Str92] for a detailed history of the problem and known algorithms).

Interestingly, the ROBDD literature already suggests an approach to widening ROBDDs that is based on prime implicants (albeit of a restricted form). Bagnara and Schachte [BS99] propose an algorithm for finding all pairs $x, y \in X$ such that $f \models x$, $f \models \neg x$ or $f \models (x \iff y)$ for some $f \in \mathbb{B}_X$. Their algorithm, which was actually devised to factorise ROBDDs, resides in $O(n^2|G|)$ where $n = |X|$. The formula $(x \iff y)$, can be decomposed into two so-called quadratic prime implicants [CH09] which hints

**Algorithm 13** WidenNPOS($f$)

**Require:** $f \in ROBDD_X$
1: $x_i \leftarrow index(f)$
2: **if** $f|_{x_i \leftarrow 0} = true \wedge f|_{x_i \leftarrow 1} = false$ **then**
3: **return** $\langle i, 0, 1\rangle :: \epsilon$
4: **else if** $f|_{x_i \leftarrow 0} = false \wedge f|_{x_i \leftarrow 1} = true$ **then**
5: **return** $\langle i, 1, 0\rangle :: \epsilon$
6: **else if** $f|_{x_i \leftarrow 0} = true$ **then**
7: **return** $\epsilon$
8: **else if** $f|_{x_i \leftarrow 1} = true$ **then**
9: **return** $\epsilon$
10: **else if** $f|_{x_i \leftarrow 0} = false$ **then**
11: **return** $\langle i, 1, 0\rangle :: \texttt{WidenNPOS}(f|_{x_i \leftarrow 1})$
12: **else if** $f|_{x_i \leftarrow 1} = false$ **then**
13: **return** $\langle i, 0, 1\rangle :: \texttt{WidenNPOS}(f|_{x_i \leftarrow 0})$
14: **end if**
15: $v_1 \leftarrow \langle i, 0, 1\rangle :: \texttt{WidenNPOS}(f|_{x_i \leftarrow 0})$
16: $v_2 \leftarrow \langle i, 1, 0\rangle :: \texttt{WidenNPOS}(f|_{x_i \leftarrow 1})$
17: **return** $\texttt{anti\_unify}(v_1, v_2)$

at the existence of an ROBDD widening. In this section we present an improved algorithm based on Plotkin's anti-unification algorithm [Plo70] for finding all pairs $x, y \in X$ such that $f \models x$, $f \models \neg x$, $f \models (x \iff y)$ or $f \models (x \iff \neg y)$ for some $f \in \mathbb{B}_X$. Although this technique is more general than the algorithm prescribed by Bagnara and Schachte [BS99], its complexity resides in $O(|G| n \log n)$ and is therefore more efficient. We denote the class of Boolean formulae expressible by such relations as $NPOS_X$. The widening algorithm, whose pseudo-code is given in Algorithm 13, takes as input an ROBDD $f$ and returns as output a set of tuples $v$ over an ordered set of variables $X = \{x_1, \ldots, x_n\}$ representing the Boolean function $\bigwedge\{f' \in NPOS_X \mid f \models f'\}$. The algorithm manipulates lists of tuples which are combined by applying anti-unification. The Boolean function $\texttt{decode}_X(v)$ explains how a list of tuples $v$ over an ordered set of variables $X = \{x_1, \ldots, x_n\}$ is interpreted as a Boolean function drawn from $NPOS_X$.

$$\texttt{decode}_X(v) = \begin{cases} \mathit{true}, & \text{if } v = \epsilon \\ x_i \wedge \texttt{decode}_X(v'), & \text{if } v = \langle i, 1, 0\rangle :: v' \\ \neg x_i \wedge \texttt{decode}_X(v'), & \text{if } v = \langle i, 0, 1\rangle :: v' \\ \texttt{decode}_X((\langle i, A, B\rangle :: v')[A \mapsto 0, B \mapsto 1]) \vee & \\ \texttt{decode}_X((\langle i, A, B\rangle :: v')[A \mapsto 1, B \mapsto 0]), & \text{if } v = \langle i, A, B\rangle :: v' \end{cases}$$

**Example 5.2.** *If* $X = \{x_1, \ldots, x_3\}$ *and* $v = \langle 1, A, B\rangle :: \langle 2, A, B\rangle :: \langle 3, 0, 1\rangle$ *then* $\texttt{decode}_X(v) = \texttt{decode}_X(\langle 1, 0, 1\rangle :: \langle 2, 0, 1\rangle :: \langle 3, 0, 1\rangle) \vee \texttt{decode}_X(\langle 1, 1, 0\rangle :: \langle 2, 1, 0\rangle :: \langle 3, 0, 1\rangle) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3) = (x_1 \iff x_2) \wedge \neg x_3$.

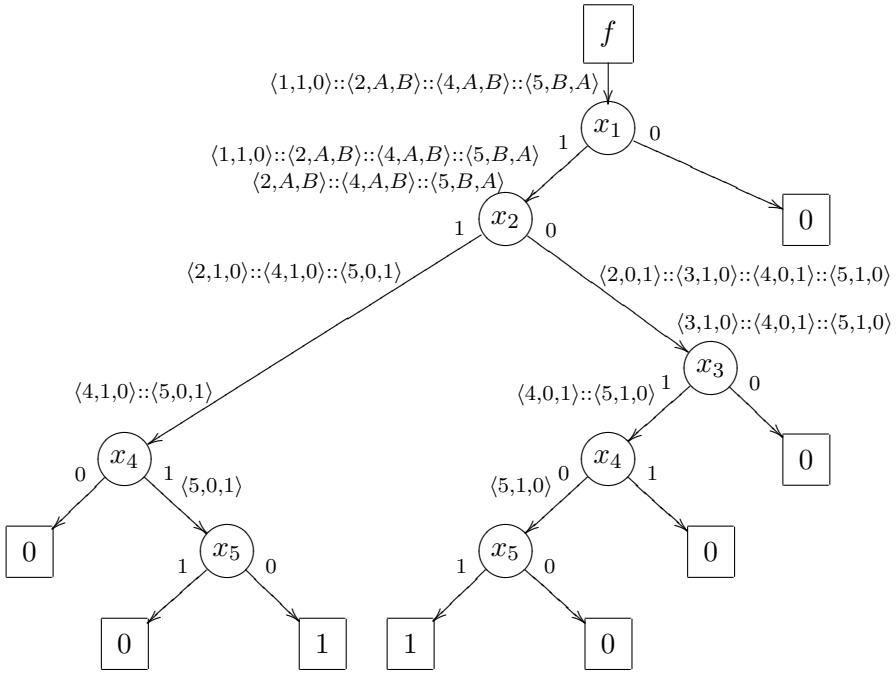


Figure 11: Algorithm 13 applied to $f = x_1 \wedge (x_2 \vee x_3) \wedge (x_2 \iff x_4) \wedge (x_4 \iff \neg x_5)$

The rationale for encoding formulae as lists of tuples is illustrated by considering two formulae $f_1, f_2 \in NPOS_X$. Suppose $f_1$ and $f_2$ are represented by $v_1$ and $v_2$, that is, $\texttt{decode}_X(v_i) = f_i$. Suppose too that the tuples are ordered lexigraphically and $v_1'$ and $v_2'$ are obtained from $v_1$ and $v_2$ by removing the tuple for any variable index which arises in one list. The anti-unification of $v_1'$ and $v_2'$ is a list of tuples $v$ such that $\texttt{decode}_X(v) = f$ where $f$ is the strongest formula in $NPOS_X$ such that $f_1 \vee f_2 \models f$. In Algorithm 13, $\texttt{anti\_unify}(v_1, v_2)$ denotes the combined filtering and anti-unification algorithm.

**Example 5.3.** *Suppose* $f_1 = x_2 \wedge x_4 \wedge \neg x_5$ *and* $f_2 = \neg x_2 \wedge x_3 \wedge \neg x_4 \wedge x_5$. *Then* $\texttt{decode}_X(v_1) = f_1$ *and* $\texttt{decode}_X(v_2) = f_2$ *where* $v_1 = \langle 2, 1, 0\rangle :: \langle 4, 1, 0\rangle :: \langle 5, 0, 1\rangle$ *and* $v_2 = \langle 2, 0, 1\rangle :: \langle 3, 1, 0\rangle :: \langle 4, 0, 1\rangle :: \langle 5, 1, 0\rangle$. *Moreover,* $\texttt{anti\_unify}(v_1, v_2) = \langle 2, A, B\rangle :: \langle 4, A, B\rangle :: \langle 5, B, A\rangle$ *and* $\texttt{decode}_X(\langle 2, A, B\rangle :: \langle 4, A, B\rangle :: \langle 5, B, A\rangle) = (x_2 \iff x_4) \wedge (x_4 \iff \neg x_5)$ *as required.*

The following example illustrates the execution of the widening algorithm for an ROBDD representing the Boolean function $f = (x_1 \wedge (x_2 \vee x_3)) \wedge (x_2 \iff x_4) \wedge (x_4 \iff \neg x_5)$.

**Example 5.4.** *A run of Algorithm 13 is illustrated for the function* $f$ *in Figure 11, for simplicity, complement edges are omitted and constant nodes not reduced. The algorithm proceeds bottom-up, computing formulae for the nodes in the ROBDD in the following order:*

| | |
|---|---|
| *leftmost* $x_5$ | $\neg x_5$ |
| *leftmost* $x_4$ | $x_4 \wedge \neg x_5$ |
| *rightmost* $x_5$ | $x_5$ |
| *rightmost* $x_4$ | $\neg x_4 \wedge x_5$ |
| $x_3$ | $x_3 \wedge \neg x_4 \wedge x_5$ |
| $x_2$ | $(x_2 \iff x_4) \wedge (x_4 \iff \neg x_5)$ |
| $x_1$ | $x_1 \wedge (x_2 \iff x_4) \wedge (x_4 \iff \neg x_5)$ |

The following lemmas build towards the proof of correctness for Algorithm 13 given in Proposition 5.4.

**Lemma 5.1.** *Given an ROBDD $f$ labelled with a variable $x_i \in X$ then $f \models g$ for some $g \in \mathbb{B}_X$ iff $(\neg x_i \wedge f|_{x_i \leftarrow 0}) \models g$ and $(x_i \wedge f|_{x_i \leftarrow 1}) \models g$.*

*Proof.* $f \models g$ iff, $(\neg x_i \wedge f|_{x_i \leftarrow 0}) \vee (x_i \wedge f|_{x_i \leftarrow 1}) \models g$ iff $\neg x_i \wedge f|_{x_i \leftarrow 0} \models g$ and $x_i \wedge f|_{x_i \leftarrow 1} \models g$. □

**Lemma 5.2.** *Given an ROBDD $f$ then $f \models x_i$ (resp. $f \models \neg x_i$) for some $x_i \in X$ iff every path from the root of $f$ to true visits a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} = false$ (resp. $g|_{x_i \leftarrow 1} = false$).*

*Proof.* For brevity we only consider the $f \models x_i$ case; the $f \models \neg x_i$ case is analogous.

- $\Leftarrow$ We first prove the *if* direction by contrapositive. Hence, assume there exists a path from the root of $f$ to *true* that does not visit a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} = false$. Observe, we have two cases:
  - firstly, the path does not visit a node labelled $x_i$. Thus there exists some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-i-1}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2) = f(\mathbf{b}_1, 1, \mathbf{b}_2) = 1$. Hence $f \not\models x_i$, a contradiction.
  - secondly, the path does visit a node $g$ labelled $x_i$, however, $g|_{x_i \leftarrow 0} \neq false$. Therefore there exists $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-i-1}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2) = 1$. Hence $f \not\models x_i$, a contradiction.
- $\Rightarrow$ To prove the *only-if* direction. By assumption, all paths from the root of $f$ to *true* visit a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} = false$. Therefore, there does not exist any $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-i-1}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2) = 1$. Thus $f \models x_i$. □

**Lemma 5.3.** *Given an ROBDD $f$ then $f \models (x_i \iff x_j)$ (resp. $f \models (x_i \iff \neg x_j)$) for some $x_i, x_j \in X$ iff every path from the root of $f$ to true visits a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} \models \neg x_j$ and $g|_{x_i \leftarrow 1} \models x_j$ (resp. $g|_{x_i \leftarrow 0} \models x_j$ and $g|_{x_i \leftarrow 1} \models \neg x_j$).*

*Proof.* For brevity we only consider the $f \models (x_i \iff x_j)$ case;

$\Leftarrow$ We first prove the *if* direction by contrapositive. Hence, assume there exists a path from the root of $f$ to *true* that does not visit a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} \models \neg x_j$ and $g|_{x_i \leftarrow 1} \models x_j$. Observe, we have two cases:

- firstly, the path does not visit a node labelled $x_i$. Thus, there exists some $\mathbf{b}_1 \in \mathbb{B}^{i-1}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-i-1}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2) = f(\mathbf{b}_1, 1, \mathbf{b}_2) = 1$. Let $g = f(\mathbf{b}_1)$, then $g|_{x_i \leftarrow 0} = g|_{x_i \leftarrow 1}$. Since $g \neq \mathit{false}$ it follows that $g|_{x_i \leftarrow 0} \models \neg x_j$ and $g|_{x_i \leftarrow 1} \models x_j$ cannot hold together.
- secondly, the path does visit a node $g$ labelled $x_i$ and $g|_{x_i \leftarrow 0} \not\models \neg x_j$ or $g|_{x_i \leftarrow 1} \not\models x_j$. Assume for instance, $g|_{x_i \leftarrow 0} \not\models \neg x_j$. But $g|_{x_i \leftarrow 0} \neq \mathit{false}$ hence by Lemma 5.2 either:
  - the path does not visit a node $h$ labelled $x_j$. Thus there exists $\mathbf{b}_1 \in \mathbb{B}^{j-i}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-j-1}$ such that $g|_{x_i \leftarrow 0}(\mathbf{b}_1, 0, \mathbf{b}_2) = g|_{x_i \leftarrow 0}(\mathbf{b}_1, 1, \mathbf{b}_2) = 1$. Specifically, $g(0, \mathbf{b}_1, 1, \mathbf{b}_2) = 1$, hence $f \not\models (x_i \iff x_j)$, a contradiction.
  - the path visits a node $h$ labelled $x_j$ but $h|_{x_j \leftarrow 1} \neq \mathit{false}$. Therefore there exists $\mathbf{b}_1 \in \mathbb{B}^{j-i}$ and $\mathbf{b}_2 \in \mathbb{B}^{n-j-1}$ such that $g|_{x_i \leftarrow 0}(\mathbf{b}_1, 1, \mathbf{b}_2) = 1$. Specifically, $g(0, \mathbf{b}_1, 1, \mathbf{b}_2) = 1$, hence $f \not\models (x_i \iff x_j)$, a contradiction.

  The $g|_{x_i \leftarrow 1} \not\models x_j$ case follows analogously.

$\Rightarrow$ To prove the *only-if* direction. By assumption, all paths from the root of $f$ to *true* visit a node $g$ labelled $x_i$ such that $g|_{x_i \leftarrow 0} \models \neg x_j$ and $g|_{x_i \leftarrow 1} \models x_j$. Therefore by two applications of Lemma 5.2 it follows that there does not exist any $\mathbf{b}_1 \in \mathbb{B}^{i-1}, \mathbf{b}_2 \in \mathbb{B}^{j-i}$ and $\mathbf{b}_3 \in \mathbb{B}^{n-j-1}$ such that $f(\mathbf{b}_1, 0, \mathbf{b}_2, 1, \mathbf{b}_3) = 1$ or $f(\mathbf{b}_1, 1, \mathbf{b}_2, 0, \mathbf{b}_3) = 1$. Thus $f \models (x_i \iff x_j)$. □

**Proposition 5.4.** *Given an ROBDD $f$ then Algorithm 13 computes a list of tuples $v$ such that* $\texttt{decode}_X(v) = f'$ *where $f' \in NPOS_X$ and $f \models f'$.*

*Proof.* We proceed by induction on the input ROBDD $f$. We consider the following five cases in order:

- $f|_{x_i \leftarrow 0} = \mathit{true}$ and $f|_{x_i \leftarrow 1} = \mathit{false}$: By Lemma 5.2 there can exist no $x_j \in X$ such that $f \models x_j$ (or $f \models \neg x_j$ analogously) for all $j > i$. Furthermore, by Lemma 5.3 there can exist no $x_j, x_k \in X$ such that $f \models (x_j \iff x_k)$ (or $f \models (x_j \iff \neg x_k)$ analogously) for all $k > j > i$. Hence, $\epsilon$ denotes *true*. However, observe $f \models \neg x_i$, hence let $\langle i, 0, 1 \rangle :: \epsilon$ denote variable $x_i = \mathit{false}$ for all $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$.
- $f|_{x_i \leftarrow 0} = \mathit{false}$ and $f|_{x_i \leftarrow 1} = \mathit{true}$: analogous to the above case.
- $f|_{x_i \leftarrow 0} = \mathit{true}$ or $f|_{x_i \leftarrow 1} = \mathit{true}$: By Lemma 5.2 there can exist no $x_j \in X$ such that $f \models x_j$ (or $f \models \neg x_j$ analogously) for all $j > i$. Furthermore, by Lemma 5.3

there can exist no $x_j, x_k \in X$ such that $f \models (x_j \iff x_k)$ (or $f \models (x_j \iff \neg x_k)$ analogously) for all $k > j > i$. Hence, $\epsilon$ denotes *true*.

- $f|_{x_i \leftarrow 0} = \mathit{false}$ or $f|_{x_i \leftarrow 1} = \mathit{false}$: By Lemma 5.2, if $f|_{x_i \leftarrow 0} = \mathit{false}$ then $f \models x_i$ in the node $f$ further if $f|_{x_i \leftarrow 1} = \mathit{false}$ then $f \models \neg x_i$ in the node $f$. Hence, if $f|_{x_i \leftarrow 0} = \mathit{false}$ then let $\langle i, 1, 0 \rangle$ denote variable $x_i = \mathit{true}$ for all $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$. Further, if $f|_{x_i \leftarrow 1} = \mathit{false}$ then let $\langle i, 0, 1 \rangle$ denote variable $x_i = \mathit{false}$ for all $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$.

- otherwise: Lemma 5.3, $f \models (x_i \iff x_j)$ for some $x_i, x_j \in X$ then $f|_{x_i \leftarrow 0} \models \neg x_j$ and $f|_{x_i \leftarrow 1} \models x_j$. Assume $f \models (x_i \iff x_j)$ then $v_1 = \langle i, 0, 1 \rangle :: \ldots :: \langle j, 0, 1 \rangle$ and $v_2 = \langle i, 1, 0 \rangle :: \ldots :: \langle j, 1, 0 \rangle$ hence $\texttt{anti\_unify}(v_1, v_2) = \langle i, A, B \rangle :: \ldots :: \langle j, A, B \rangle$. Let $\langle i, A, B \rangle :: \ldots :: \langle j, A, B \rangle$ denote $x_i = \mathit{true} \wedge x_j = \mathit{true}$ or $x_i = \mathit{false} \wedge x_j = \mathit{false}$ for all $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$. Further, let $f \models (x_i \iff \neg x_j)$ then $v_1 = \langle i, 0, 1 \rangle :: \ldots :: \langle j, 1, 0 \rangle$ and $v_2 = \langle i, 1, 0 \rangle :: \ldots :: \langle j, 0, 1 \rangle$ hence $\texttt{anti\_unify}(v_1, v_2) = \langle i, A, B \rangle :: \langle j, B, A \rangle$. Let $\langle i, A, B \rangle :: \ldots :: \langle j, B, A \rangle$ denote $x_i = \mathit{true} \wedge x_j = \mathit{false}$ or $x_i = \mathit{false} \wedge x_j = \mathit{true}$ for all $\mathbf{b} \in \mathbb{B}^n$ such that $\mathbf{b} \in \mathit{model}_X(f)$. Finally, let $v_1 = \langle i, A, B \rangle$ and $v_2 = \langle i, A, B \rangle$ then $\texttt{anti\_unify}(v_1, v_2) = \langle i, A, B \rangle$.

By applying Lemma 5.1 inductively to each node in the ROBDD $f$ where all nodes are one of the above cases, the result follows. □

The overall complexity of the algorithm is in $O(|G| n \log n)$, since with the exception of $\texttt{anti\_unify}(v_1, v_2)$, all steps in Algorithm 13 require constant time. The `anti_unify` operation requires $O(n \log n)$ time for lists possessing at most $n$ tuples when Plotkin's anti-unification algorithm [Plo70] is implemented using an AVL tree. Since each node is visited *at most* once, the overall complexity of Algorithm 13 is in $O(|G| n \log n)$.

The value of this widening, is not only its tractability, but that it can only admit chains of linear length. This follows from the observation that the number of truth assignments for any $f \in NPOS_X$ is a power of 2. The strongest element of $NPOS_X$ is the Boolean function $\wedge X$ which has precisely 1 truth assignment. Conversely, the weakest element is the Boolean function *true* which possesses $2^n$ models. It follows that chain length cannot exceed $n + 1$. This widening therefore offers a compromise between efficiency (i.e. provably polynomial complexity) and expressiveness, that suits analyses which only require the manipulation of Boolean dependencies [HACK00].

## 5.9 Conclusions

In this chapter we have proposed a new widening for Boolean formulae represented as ROBDDs and an algorithm for realising it. The widening can be used to either bound the number of times that an ROBDD is updated in an iterative abstract interpretation

based program analysis, or approximate an ROBDD with another that has a more space-efficient representation. Empirical evidence suggests that the widening is both useful and surprisingly tractable. Furthermore, the widening exhibits the diminishing property in that approximations rapidly converge on to the optimal solution as the length of prime implicants is relaxed. Moreover, unlike previous proposed approximation techniques, the widening is not dependant on the underlying variable ordering. In addition to this widening, the chapter has presented a provably polynomial widening technique that only admits chains of length linear in the number of propositional variables.

# Chapter 6

# Widening ROBDDs Randomly

"Anyone who considers arithmetical methods of producing
random digits is, of course, in a state of sin."
- J. V. Neumann

**Abstract.** In this chapter we investigate the approximation of Boolean formulae represented as ROBDDs using randomised techniques. The key idea is to forfeit the guarantee of finding all prime implicants of minimal length, in order to obtain a tractable widening. In particular, random selection is applied in order to find a short prime implicant with high probability. By iterating this tactic, a collection of short primes can be quickly found which leads to surprisingly accurate approximation. The experimental results suggest that randomisation can achieve approximations which compare with those generated by systematic prime enumeration. Moreover, the iterative nature of the randomised approach makes it naturally anytime.

## 6.1 Introduction

A randomised (or probabilistic) algorithm is an algorithm which employs a degree of randomness as part of its logic. Theoretically, randomised algorithms are formulated by augmenting a machine with some randomisation device, for instance, a Turing machine would be augmented with an auxiliary tape of random values. Practically, randomised algorithms are normally implemented with an efficient pseudo random number generator. A randomised algorithm typically utilises the pseudo random number generator in one of two ways: to randomly generate prospective solutions; or to guide non-deterministic choices in the otherwise deterministic computation of a solution.

Randomised algorithms are necessarily multipass, where a single pass produces a solution to a problem with a probability of correctness $0 < \epsilon < 1$. Thus after $n$ independent passes of the algorithm, a correct solution will be computed with probability

$1 - (1 - \epsilon)^n$. For example, consider a problem involving search, for which a random selection has $\epsilon = 1/2$ probability of finding the best solution according to some optimisation criteria. Then, after $n = 10$ independent random selections, the probability of obtaining the optimal solution is $1 - 1/1024 \approx 0.999$.

Randomised algorithms have proven useful in solving problems for which a computationally feasible algorithm is unknown. Randomisation has proven particularly powerful when solving number theoretic problems such as primality testing and prime factoring as exemplified by the Rabin-Miller [Rab80] and Pollard-$\rho$ [Pol75] algorithms. These algorithms are attractive because for both of these problems no efficient deterministic algorithms are known. In algorithm design, randomisation is considered a last resort [Pre88] and is usually only applied when there is little chance of devising a polynomial time solution. Furthermore, randomisation is only practical when there is sufficient structure or bias in the problem so that random selection is likely to gravitate towards a solution, rather than a non-solution. Randomisation is worthy of consideration for widening ROBDDs because of the negative result that suggests that computing a shortest prime implicant of an ROBDD is very unlikely to be polynomial. Moreover, selecting a random path from an ROBDD and then extracting the prime implicant contained within it is more likely to find an implicant that is short than long. The interesting research question, is whether this bias can be exploited to obtain accurate and efficient approximation. Randomised algorithms often inspire so-called derandomised algorithms which are deterministic algorithms that attempt to retain the conceptual simplicity of their randomised counterpart. A related research question is therefore whether a derandomised variant retains the bias necessary for efficiency.

The overall running time of a randomised algorithm is the product of the cost of a single pass and the total number of passes required. However, a single pass that is computationally lightweight might only generate an answer that has a small probability of being a solution. Thus although each pass is cheap, the number of passes would be large and hence their product would become unacceptable. In this chapter, we explore how different branch selection tactics, with different costs, effect the quality of the prime implicants found and hence the precision of the ensuing ROBDD approximation.

Although one would think that a widening based on randomisation would be fundamentally different from that presented previously, it turns out that the two widenings share much commonality in that they are both based on finding prime implicants. However, a key difference between the widening of the previous chapter and those proposed in this, is the granularity at which they operate. When each pass of a widening generates primes of successively longer length, each iteration may well contribute many primes to the approximation generated on the previous iterate. This means that the difference in precision between the approximation generated for length $k$ and $k + 1$ primes is often large. Thus, the increments in precision in this approach to prime enumeration are

potentially coarse-grained. For instance, it may well be feasible to generate all primes up to length $k = 10$, say, but practically impossible to generate all primes up to length $k = 11$. This behaviour is manifested in the rot-#149 example, as illustrated in Figure 8. Such explosive behaviour often occurs because the number of primes whose length does not exceed $k$ do not increase linearly with $k$, at least in the average case [MP64]. By way of contrast, the widening techniques proposed in this chapter add only a single prime to a monotonically increasing approximation, and therefore these widenings are potentially more attractive because the precision increment is more fine-grained.

## 6.2 Contributions

In this chapter we propose a new widening technique for Boolean functions represented as ROBDDs based upon the random generation of prime implicants. This technique leads to two widenings: one is truly random and the other is a deterministic derandomised algorithm. We summarise our contributions as follows:

- The widenings presented herein enable finer control of the computational resources required to construct the approximation. This is because the algorithms add new implicants one-by-one to the approximation rather than all the implicants of a given length in one monolithic step. A consequence of this is that little computational resource is expended between the generation of one implicant and the next which means the algorithm is truly interruptible, which is considered to be a particularly desirable feature of an anytime algorithm.

- The step of finding a single implicant is provably polynomial, though adding an implicant of maximal length $n$ to the approximation can potentially produce an ROBDD of size $O(n|G|)$ where $|G|$ is the initial size of the approximation. The problem of generating an exorbitantly large resultant approximation can be avoided merely by stopping the algorithm when $|G|$ exceeds a prescribed bound.

- Finding a single implicant does not create any intermediate ROBDDs or require the input ROBDD itself to be altered. This is important because an ROBDD needs approximation when it is already intractably large. As a consequence of the reduced number of ROBDD operations, the widening can achieve high precision with only modest computational effort. Experimental results are presented which demonstrate the tractability of randomised widening.

Despite these merits, the widenings presented in this chapter are sensitive to the variable ordering, that is, they may return different approximations for the same Boolean function if variable reordering is applied. This chapter is structured as follows: Section 6.3 outlines how randomisation can be applied in widening. Sections 6.4 and 6.5 present the experimental results and the conclusions respectively.

## 6.3 The Widening

Given a Boolean function $f \in \mathbb{B}_X$ the problem of computing a widening (an over-approximation) is essentially that of selecting a function from $F = \{f' \in \mathbb{B}_X \mid f \models f'\}$. The computational issue is that $|F| = \sum_{k=0}^{2^n-m} \binom{2^n-m}{k}$ where $m = |model_X(f)|$ and moreover, $\sum_{k=0}^{2^n-m} \binom{2^n-m}{k} = 2^{2^n-m}$ by the binomial identity [AS72, p 10]. Thus it is not practical to compute $F$ directly. Furthermore, as the size of $F$ increases, the chance of randomly selecting a good approximation from $F$, that is, an approximation with relatively few extra models, becomes increasingly small.

One would think then, that randomisation has little hope of finding accurate approximations of Boolean formulae. However, it turns out, that randomisation can be deployed to discover primes whose length are typically small. As shown in the previous chapter, given two primes $p, p' \in primes(\neg f)$ such that $|p'| < |p|$ then $p'$ is a better candidate for forming an approximation. This follows because $p'$ contributes $2^{n-|p'|}$ truth assignments to $\neg f$ whereas $p$ contributes only $2^{n-|p|}$. Thus $p'$ produces a more accurate under approximation of $\neg f$ than $p$ and hence a more accurate over approximation of $f$. The force of this observation is that short prime implicants contribute short paths in the ROBDD that constitutes the approximation. The converse to this result, is that short paths in the original ROBDD can be used to find a collection of short primes which provide the basis for constructing an accurate approximation. The following proposition builds towards this tactic by showing that each path to *true* in an ROBDD contains a unique prime implicant. The force of this result is that to find $k$ short primes, it is sufficient to examine just $k$ short paths.

**Proposition 6.1.** *Suppose a path to true in an ROBDD $f$ defines an implicant $p$. Then if $p_1, p_2 \in primes(f)$ and $p \models p_1$ and $p \models p_2$ it follows that $p_1 = p_2$.*

*Proof.* Suppose $f$ is defined over the set of variables $X = \{x_1, \ldots, x_n\}$. We proceed by induction on $n$.

- Suppose $n = 1$. Then
  - Suppose $f = false$. Then $primes(f) = \emptyset$ and there are no successful paths and also no primes.
  - Suppose $f = x_1$. Then $primes(f) = \{x_1\}$ and the unique successful path contains the unique prime.
  - Suppose $f = \neg x_1$. Then $primes(f) = \{\neg x_1\}$ and the unique successful path contains the unique prime.
  - Suppose $f = true$. Then $primes(f) = \{true\}$ and the unique successful path contains the unique prime.

- Suppose that the inductive hypothesis holds for all ROBDDs $f$ defined over $n$ variables. To extend the result to $n+1$ variables let $p_1, p_2 \in \mathit{primes}(f)$ such that $p_1 \neq p_2$. We have to show that $p_1$ and $p_2$ arise in different paths of $f$. Now $p_j = (\wedge X_j) \wedge \neg(\vee Y_j)$ for some $X_1, X_2, Y_1, Y_2 \subseteq \{x_1, \ldots, x_n, x_{n+1}\}$. Let $n_j = \max\{i \mid x_i \in X_j \cup Y_j\}$ for $1 \leq j \leq 2$.

  - Suppose $n_1 < n_2$. There is a unique path for $p_1$ through the ROBDD that ends in *true*. This path does not visit $x_{n_2}$ hence $p_2$ cannot occur in this path.
  - Suppose $n_1 > n_2$. Like the previous case.
  - Suppose $n_1 = n_2$. For brevity let $m = n_1$.
    - Suppose $p_1 \models x_m$ and $p_2 \models \neg x_m$. Then any path that contains $p_1$ cannot contain $p_2$.
    - Suppose $p_1 \models \neg x_m$ and $p_2 \models x_m$. Like the previous case.
    - Suppose $p_1 \models x_m$ and $p_2 \models x_m$. Put $p'_j = \exists_{x_m}(p_j)$ and $f' = f|_{x_m \leftarrow 1}$. Then $p'_1 \neq p'_2$ and $p'_1, p'_2 \in \mathit{primes}(f')$. But $f'$ is defined over no more than $n$ variables and thus by induction $p'_1$ and $p'_2$ arise in different paths of $f'$. It follows that $p_1$ and $p_2$ arise in different paths of $f$.
    - Suppose $p_1 \models \neg x_m$ and $p_2 \models \neg x_m$ symmetric to the previous case. □

### 6.3.1 Algorithmic Framework for Random Widening

Rather than propose a single widening, we present an algorithmic framework which can be instantiated with various randomisation techniques. The resulting algorithms are distinguished by the degree to which they are informed by random input. Each of the algorithms perform the following steps: (i) initialise $g \leftarrow \mathit{true}$ and put $f' \leftarrow \neg f$. Recall that negation can be computed in $O(1)$ time with ROBDDs supporting complement edges [BRB90]. (ii) select a short prime implicant $p$ of $f'$. This step is discussed in §6.3.2. (iii) add $\neg p$ to the accruing approximation, that is, assign $g \leftarrow g \wedge \neg p$. (iv) possibly remove $p$ from $f'$, that is, compute the function $f' \leftarrow f' \wedge \neg p$. This step is discussed in §6.3.3. Finally, repeat from step (ii) unless $g$ has reached some predefined size threshold or a timeout has expired. This algorithmic scheme results in an approximation $g$ such that $f \models g$.

### 6.3.2 Selecting an Implicant

Algorithm 14 shows how randomisation can be employed to find a short prime. The algorithm merely returns a random path to *true*, which by virtue of Proposition 6.1, must contain a single prime. Interestingly, the path is more likely to contain a short prime than a longer one. This is because the probability of randomly selecting a truth assignment is biased towards the primes of short length rather than the primes of longer

length. This follows as a consequence of a short prime possessing more models than the longer one. In fact the ratio between the number of assignments is exponential in the length difference of the two primes.

Algorithm 14 computes a random path to *true* within the ROBDD given in the parameter $f$ in $O(n)$ time. When the function is called again, a different path is likely to be chosen, hence there is no need to include control structures which ensure that the same path is not generated repeatedly.

**Algorithm 14** SelectImplicantRandom($f$)

**Require:** $f \in ROBDD_X$
1: $x_i \leftarrow index(f)$
2: **if** $f = true$ **then**
3: **return** $true$
4: **else if** $f|_{x_i \leftarrow 0} = false$ **then**
5: **return** $x_i \wedge \texttt{SelectImplicantRandom}(f|_{x_i \leftarrow 1})$
6: **else if** $f|_{x_i \leftarrow 1} = false$ **then**
7: **return** $\neg x_i \wedge \texttt{SelectImplicantRandom}(f|_{x_i \leftarrow 0})$
8: **end if**
9: **if** $\texttt{rand}(0, 1) = 0$ **then**
10: **return** $\neg x_i \wedge \texttt{SelectImplicantRandom}(f|_{x_i \leftarrow 0})$
11: **else**
12: **return** $x_i \wedge \texttt{SelectImplicantRandom}(f|_{x_i \leftarrow 1})$
13: **end if**

Since a path to *true* must contain a prime then, as Umans [Uma99] points out, one might expect a short path to contain a shorter prime than a longer path. This motivates derandomising the random selection of paths to generate paths in increasing order of size. Derandomisation attempts to trade simplicity for determinacy, that is recover determinacy at the sake of complicating the underlying data structures. In this case, derandomisation amounts to enumerating paths in increasing size by employing a priority queue data structure which supports a form of A* search [HNR68]. This motivates Algorithm 15 thats finds a path from a given initial node to a given goal node. The initial node is the root node of the ROBDD $f$ and goal the *true* node (the *true* node does not occur repeatedly in any ROBDD).

The A* algorithm computes an optimal path by maintaining a set of partial paths in a priority queue. The priority of each partial path is the sum of the length of the partial path and the shortest distance from the end of the path to the goal node. In Algorithm 15 the priorities $d + d_0$ and $d + d_1$ estimate the distance along the partial path $p$ through the negative and positive co-factors to the goal node. When computing these priorities, the values $d_0$ and $d_1$ can be precomputed prior to invoking the search algorithm. This preprocessing can be performed in time linear in the size of the ROBDD $f$ and the result cached for every sub-ROBDD of $f$. Actually, CUDD [Som05] provides

the functionality to compute the shortest implicant of an ROBDD as part of its machinery to compute small DNF covers, but lacks the capability to enumerate implicants of successively larger length. A* provides a mechanism for such enumeration.

**Algorithm 15** SelectImplicantA*$(f, pq)$

**Require:** $f \in ROBDD_X$
1: **loop**
2: $\langle f, p, d\rangle \leftarrow pq$.`remove`()
3: **if** $f = true \vee f = false$ **then**
4: **if** $f = true$ **then**
5: **return** $\langle p, pq\rangle$
6: **end if**
7: **else**
8: $x_i \leftarrow index(f)$
9: $d_0 \leftarrow$ `true_distance`$(f|_{x_i \leftarrow 0})$
10: $d_1 \leftarrow$ `true_distance`$(f|_{x_i \leftarrow 1})$
11: $pq$.`insert`$(d + d_0, f|_{x_i \leftarrow 0}, (p \wedge \neg x_i), d + 1)$
12: $pq$.`insert`$(d + d_1, f|_{x_i \leftarrow 1}, (p \wedge x_i), d + 1)$
13: **end if**
14: **end loop**

The first time `SelectImplicantA*`$(f, pq)$ is called, the priority queue $pq$ is initialised to contain the single vacuous path *true* that is assigned the priority of 0. Note that this value is inconsequential, since there are no other paths in the priority queue. The function `true_distance`$(f)$ merely looks up the precomputed distance from a node $f$ to the goal node *true*. As well as returning a path, the function also returns the updated priority queue, which is passed to the next call to the function.

### 6.3.3 Computing and Removing a Prime Implicant

Once a path $p$ is found in the ROBDD $f'$, then it is necessary to extract a prime implicant $p'$ such that $p \models p' \models f'$. To compute the prime implicant $p'$ from the implicant $p$, we attempt to remove variables from $p$ by using existential quantification to obtain $p'$ whilst retaining the entailment relationship $p' \models f'$. An interesting refinement of this scheme is that the variable with the highest index in the implicant $p$ must occur in the prime implicant $p'$ contained within it. This is formally stated in the following lemma.

**Lemma 6.1.** *Given an ROBDD $f$ and a path $p$ of $f$ such that $p \models p' \models f$, then $p'$ contains the variable of $p$ with the highest index.*

*Proof.* Let the variable with highest index in $p$ be $x_i$. Further, let $g$ be the node labelled $x_i$ on the path $p$. However, since either $g|_{x_i \leftarrow 0} = true$ or $g|_{x_i \leftarrow 1} = true$, but not both, it follows that $\exists_{x_i}(p) \not\models f$ since if $\exists_{x_i}(p) = p$ then the node $g$ would be reduced. □

The algorithm listed below, extracts a prime from an implicant $p$ by considering each variable in $p$ with the exception of $x_{i_m}$ which, by Lemma 6.1, is essential.

**Algorithm 16** ImplicantToPrime$(f, p)$

**Require:** $f, p \in ROBDD_X$
1: $\{x_{i_1}, \ldots, x_{i_m}\} \leftarrow var(p)$
2: **for each** $1 \leq j < m$ **do**
3: $p' \leftarrow \exists_{x_{i_j}}(p)$
4: **if** $p' \models f$ **then**
5: $p \leftarrow p'$
6: **end if**
7: **end for**
8: **return** $p$

Once a prime has been extracted from the ROBDD $f'$, we may choose to employ *deletion*, that is to remove, $p'$ from $f'$ by replacing $f'$ with $f' \wedge \neg p'$. This avoids $p'$ being generated repeatedly and ensures that $g$ strictly increases in precision when a prime is inserted. However, there is a computational overhead associated with the approach, since $f'$ is no longer static.

## 6.4 Experimental Results

In order to investigate the applicability of randomisation techniques in ROBDD approximation, both the randomised and derandomised widenings proposed in this chapter have been implemented using the Colorado University Decision Diagram (CUDD) package [Som05]. In the spirit of the previous chapter, and to provide a basis for comparison, the Boolean formulae used to assess the widenings coincide with those used in Chapter 5, and can be found in Table 11.

To compare the randomised and derandomised variants of the widenings, the effect of deletion, and the rate of convergence against time, we present a series of four figures each containing eight graphs, one graph for each Boolean function. These graphs are given in Figures 12, 13, 14 and 15. The graphs plot the rate of convergence of the approximation, showing the precision of the approximation against the cumulative time spent widening. The precision of the approximation is measured as the ratio of the number of truth assignments in the input ROBDD against the number of truth assignments in the approximation constructed from the prime implicants: the so-called minterm ratio. One desirable property of a randomised algorithm is predictability, which in this case, requires the converge rate to be comparable over multiple independent runs. Thus, experiments utilising randomised prime implicant generation were performed a total of 64 times using different seed values for the random number generators.

Table 13 summarises the data presented in the graphs at the time points of 16 and

| ID | with deletion | | | | without deletion | | | |
|---|---|---|---|---|---|---|---|---|
| | Randomised | | Derandomised | | Randomised | | Derandomised | |
| | 16 | 32 | 16 | 32 | 16 | 32 | 16 | 32 |
| 1. | 1.0327 | 1.0270 | 1.0558 | 1.0449 | 1.0259 | 1.0213 | 1.0255 | 1.0224 |
| 2. | 1.0504 | 1.0439 | 1.0673 | 1.0552 | 1.0347 | 1.0292 | 1.0318 | 1.0299 |
| 3. | 1.0313 | 1.0226 | 1.1654 | 1.1304 | 1.0582 | 1.0541 | 1.0576 | 1.0500 |
| 4. | 1.0758 | 1.0587 | 1.1872 | 1.1694 | 1.1029 | 1.0930 | 1.0772 | 1.0770 |
| 5. | 1.0059 | 1.0046 | 1.2814 | 1.1978 | 1.0049 | 1.0030 | 1.0060 | 1.0060 |
| 6. | 1.0059 | 1.0044 | 1.2795 | 1.0953 | 1.0053 | 1.0048 | 1.0066 | 1.0066 |
| 7. | 1.0071 | 1.0034 | 1.0023 | 1.0017 | 1.0254 | 1.0184 | 1.0308 | 1.0308 |
| 8. | 1.0040 | 1.0019 | 1.0010 | 1.0009 | 1.0076 | 1.0053 | 1.0185 | 1.0185 |

Table 13: Minterm ratio with and without deletion after 16 and 32 seconds.

32 seconds. The **ID** column identifies the benchmark circuit from Table 11 given in the previous chapter. The remainder of the table is separated into two sections for experiments performed with deletion and without deletion. Each entry in the table gives the minterm ratio, that is, the precision of the approximation obtained, at the two time points for both the randomised and derandomised algorithms.

Observe that the effect of deletion is small which suggests that its complexity is not repaid in terms of approximation accuracy. There is also little difference between the approximations obtained by the randomised and derandomised widenings when deletion is not applied. Thus, if the primary desire was simplicity of implementation, then the randomised algorithm is more attractive. Conversely, if the requirement was predictability, then the derandomised version of the widening would be more desirable.

One might expect an unprincipled randomised approach to produce erratic and unpredictable results. However, the empirical evidence presented in Figures 12, and 14 show this is not that case. In fact, there is a surprising degree of correlation between the 64 runs of the randomised algorithm, with the exception of the fifth and sixth graphs of Figure 14. Interestingly, these circuits possess just two implicants of length 3 with the vast majority of the primes exceeding 5 in length. This unusual distribution of primes may explain the variance in the convergence rates. Quite apart from being naturally anytime, both the randomised and derandomised approaches both possess the diminishing property in that the approximations they deliver converge rapidly.

It is interesting to see that the profiles of many of the graphs in Figures 14 and 15 are almost identical. (This can be seen by holding copies of the figures back-to-back.) The differences in the remaining graphs stem from the start-up overheads of the A* algorithm. For the very large circuits, the time required to compute the shortest distances to *true* is not insignificant. As a consequence, the randomised widening produces better approximations early in the search, but the derandomised widening achieves better

convergence over the long term.

In order to provide a meaningful comparison of the randomised and derandomised variants of the widening presented in this chapter with the widening presented in Chapter 5 we present a series of graphs, one graph for each Boolean function; these are given in Figure 16. In keeping with graphs given earlier in this section, the graphs plot the rate of convergence of the approximation, showing the precision of the approximation against the cumulative time spent widening for each of the widenings presented in this chapter and the $\nabla_k$ widening presented in Chapter 5. In each of the graphs, the derandomised widenings are denoted DS - Derandomised Selection with (DS-D) and without (DS-ND) Deletion, the randomised widenings are denoted RS-D and RS-ND likewise. Finally, the widening presented in Chapter 5 is denoted widen-k.

As can be seen from the graphs of Figure 16, the $\nabla_k$ widening is competitive with both the derandomised and randomised variants. In the case of both the *pair* (graphs $1, 2$) and *mm9b* (graphs $3, 4$) circuits, the $\nabla_k$ widening achieves convergence faster than all the derandomised and randomised widening variants. However, the $\nabla_k$ widening does not achieve convergence faster across all benchmark circuits, indeed, for both of the *rot-#149* (graph 7) and *rot-#172* (graph 8) circuits the $\nabla_k$ widening fails to converge upon the solution at a rate comparable to all the derandomised and randomised widening variants.

## 6.5 Conclusions

In this chapter we have presented a widening for Boolean functions represented as ROBDDs based on the random enumeration of prime implicants. The widening randomly selects a path through an ROBDD and then deterministically extracts the prime implicant contained within it. The prime implicant is then added to the accruing approximation. The simplicity of the widening makes it attractive for implementation, and yet it is still capable of generating accurate approximations efficiently. The widening is naturally anytime, and is truly interruptible in the sense that the time divisions between the generation of one prime and the next are small.

The desire to discover short primes led to a derandomised variant of the algorithm based on A* search. This variant incrementally searches short paths in the ROBDD as a tactic for discovering short primes. The resulting widening is more predictable than that based on randomisation but the widening does have a non-trivial startup cost which is significant when widening very large ROBDDs.

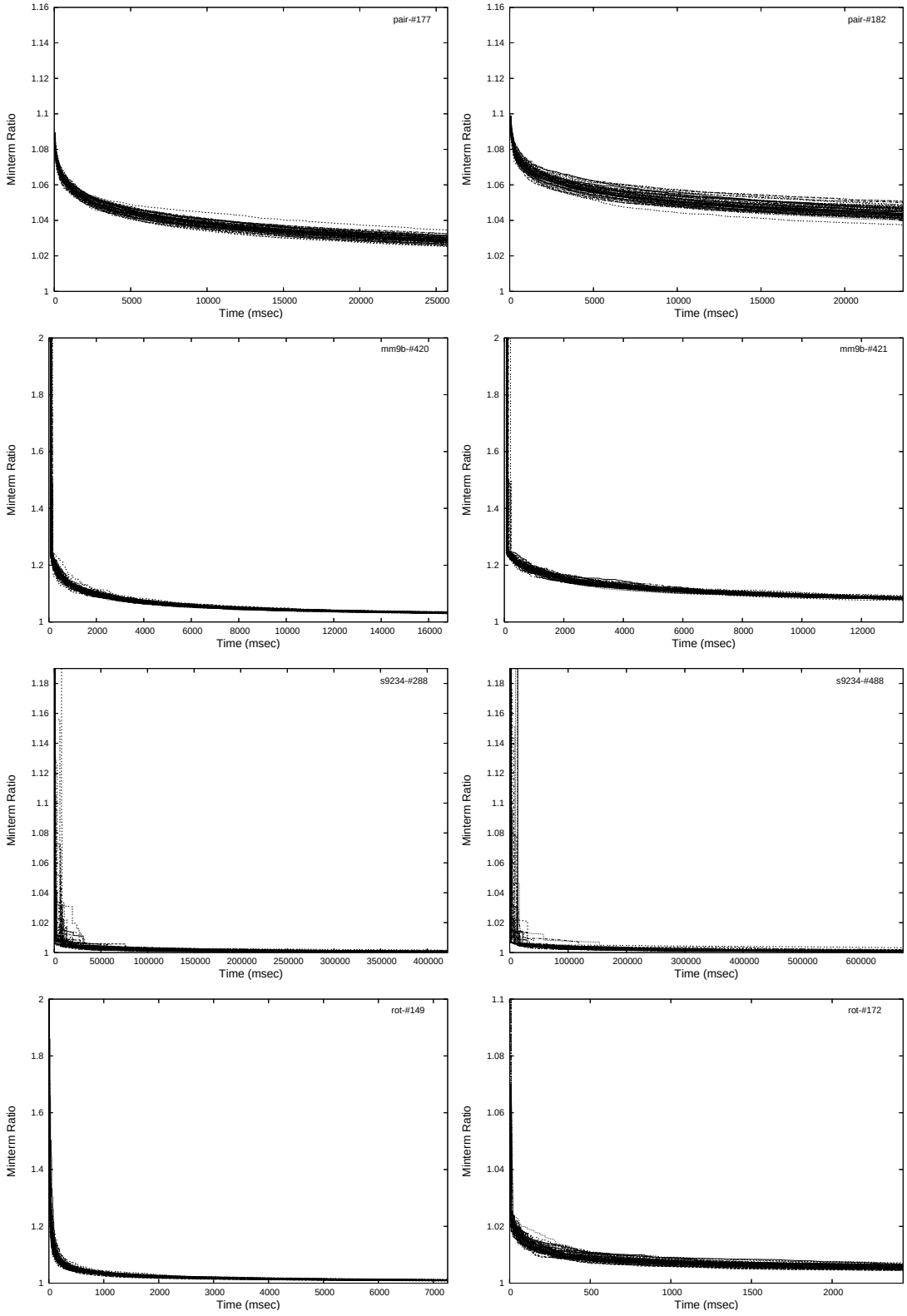


Figure 12: Random Selection with Deletion: Minterm ratio against time in milliseconds.

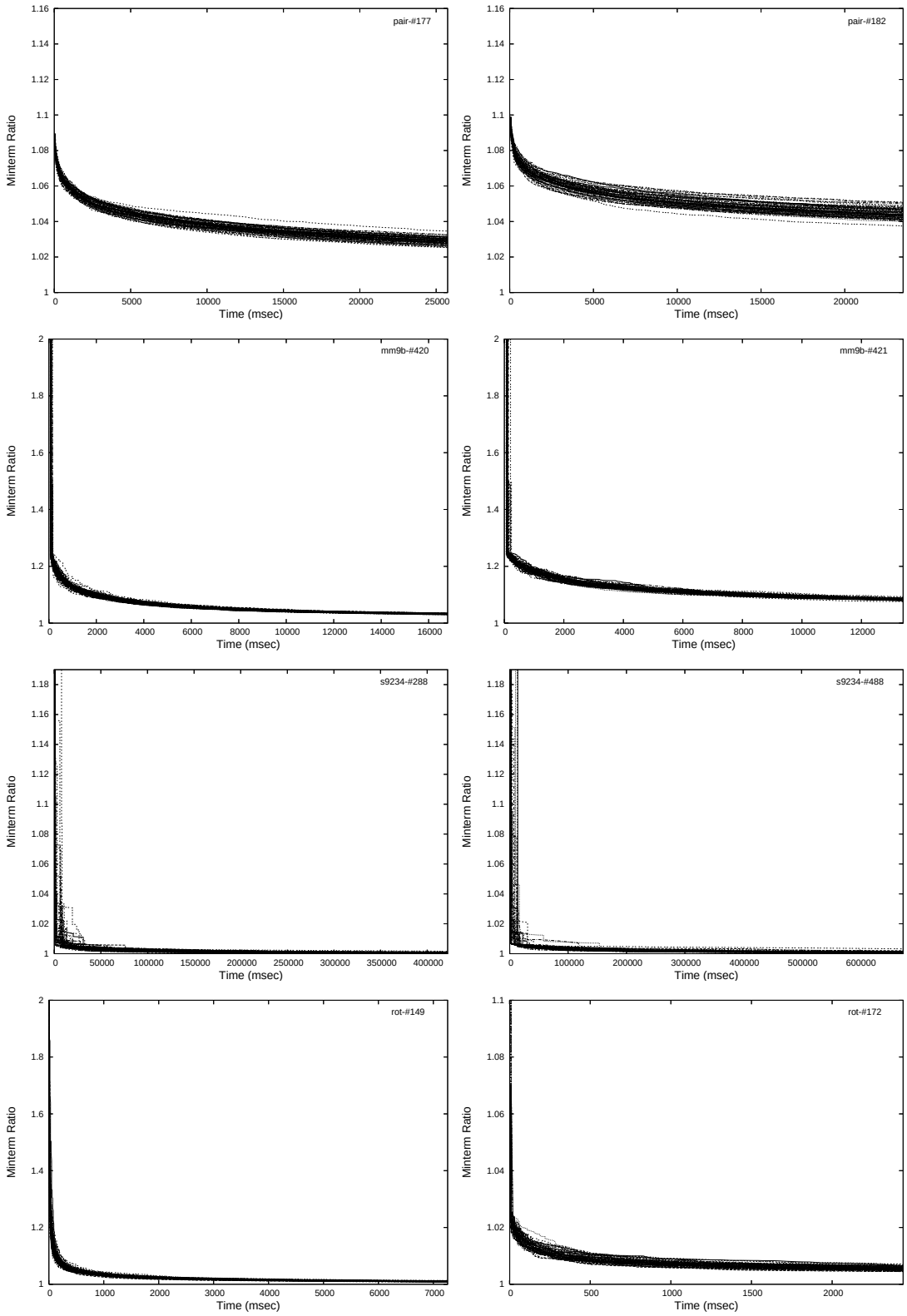


Figure 13: Derandomised Selection with Deletion: Minterm ratio against time in milliseconds.

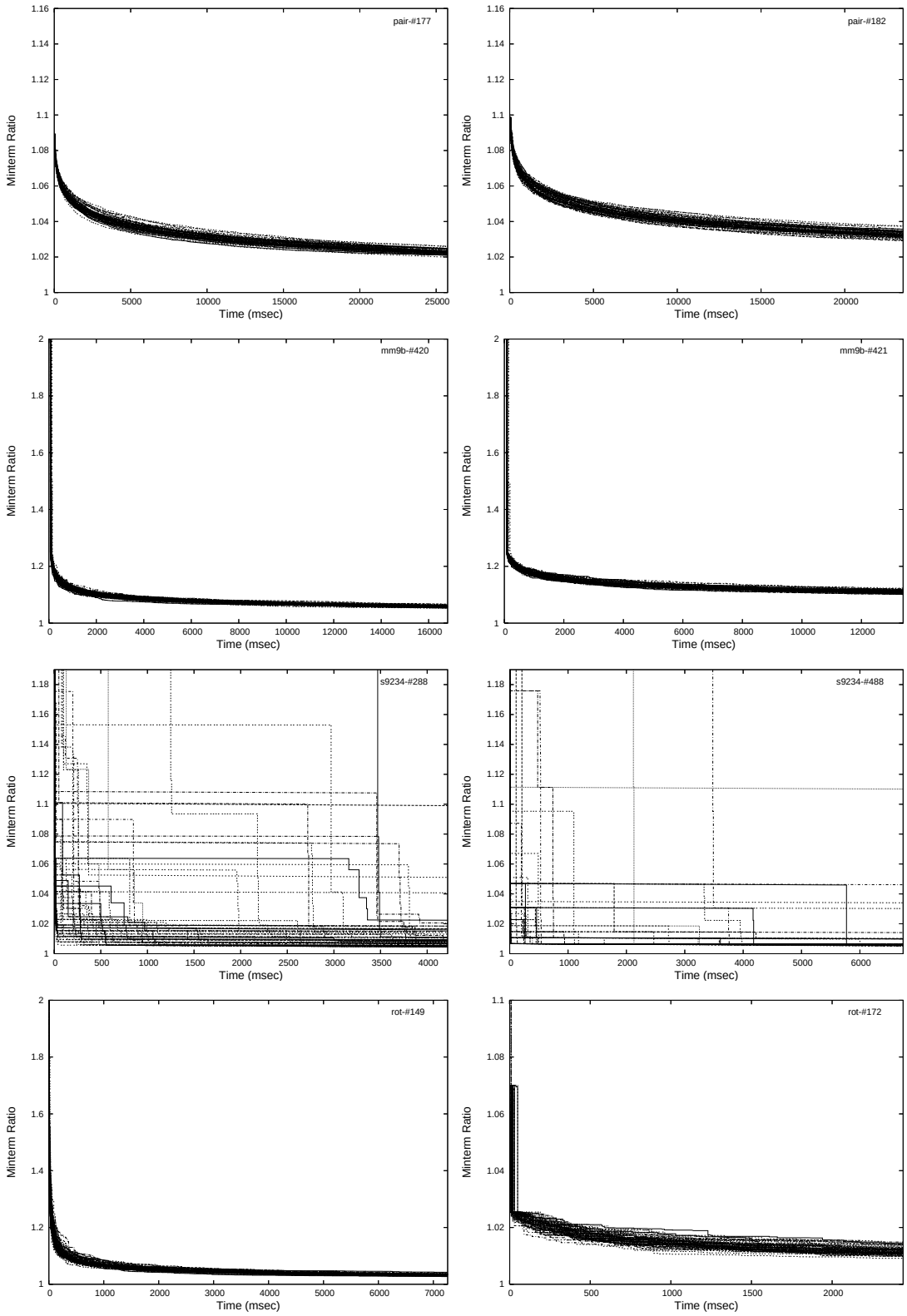


Figure 14: Random Selection without Deletion: Minterm ratio against time in milliseconds.

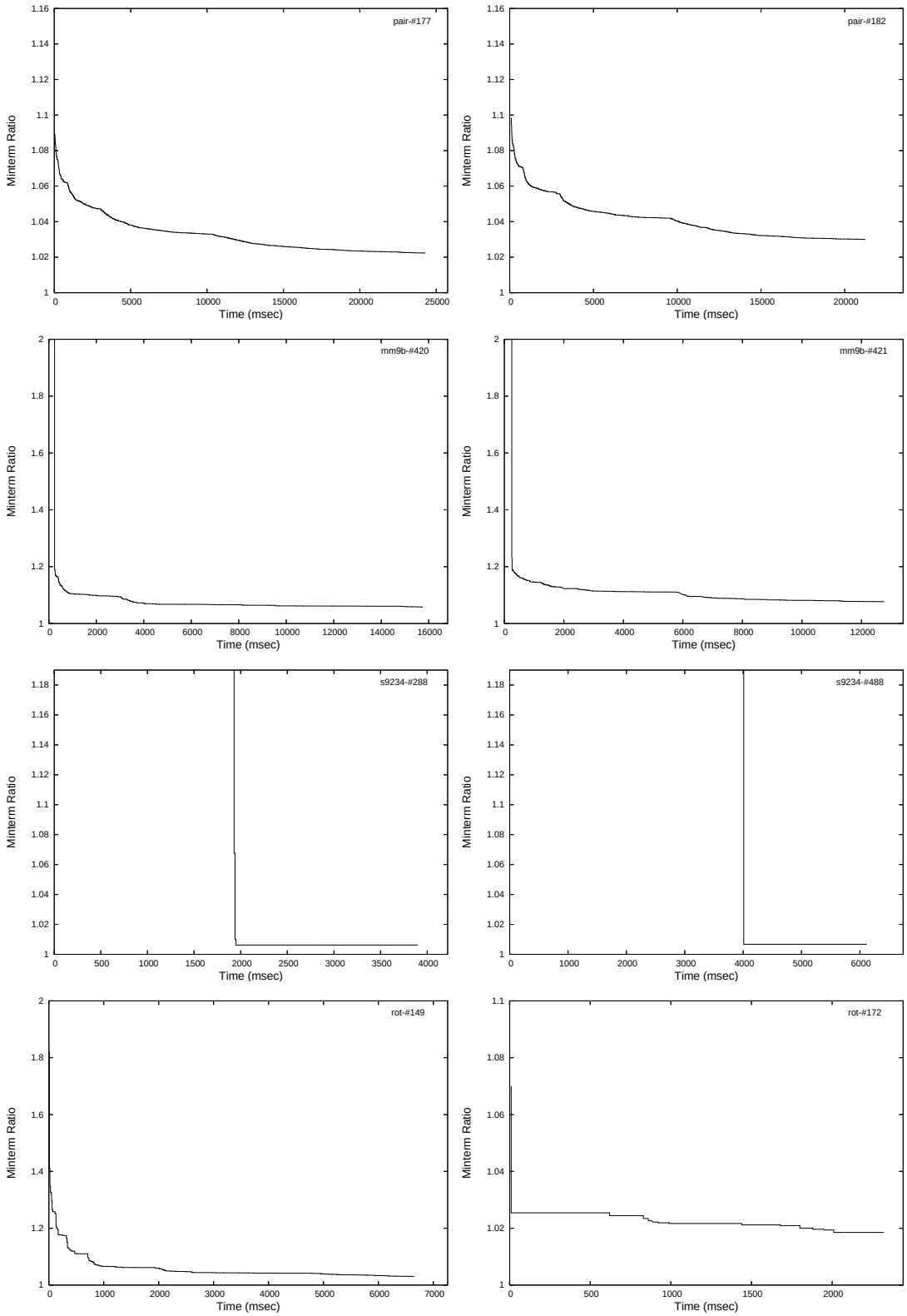


Figure 15: Derandomised Selection without Deletion: Minterm ratio against time in milliseconds.

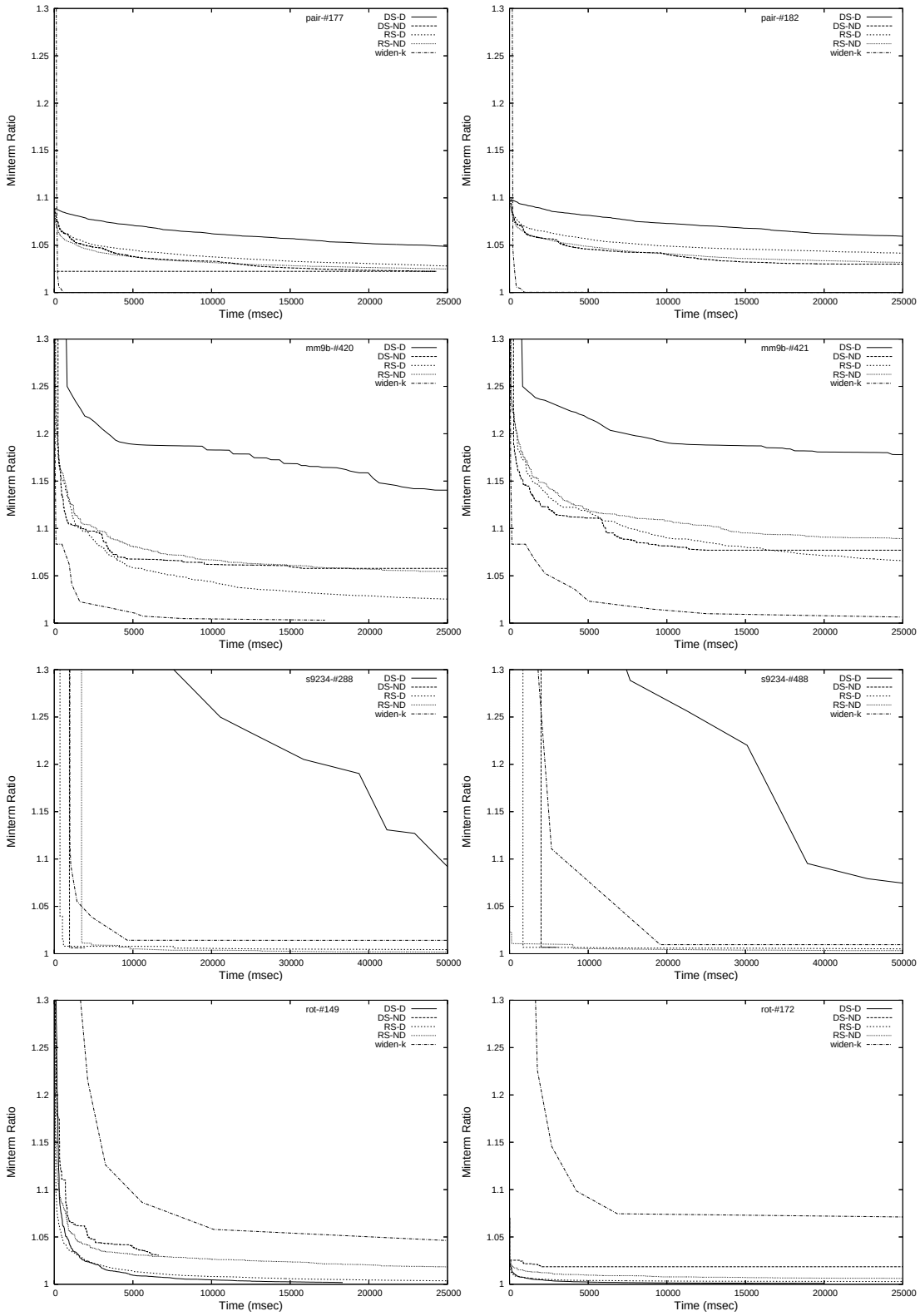


Figure 16: Randomised widening comparison: Minterm ratio against time in milliseconds.

# Chapter 7

# Conclusion

"When small men begin to cast big shadows,
it means that the sun is about to set."
- L. Yutang

## 7.1 Summary of Contributions

In this thesis we have proposed a family of algorithms for solving the symmetry detection and approximation problems with application to the ROBDD data structure. We have shown how the anytime concept is useful in the context of ROBDD manipulation. Surprisingly, despite the generality of anytime computation, the overhead involved appears to be minimal in the case of symmetry detection and approximation. Further, since symmetry computation and approximation can both be applied to reduce the size of an ROBDD, both techniques find application in the context of logic synthesis, program analysis and indeed any application requiring the manipulation of sufficiently small ROBDDs. With a view to the future, the iterative nature of the anytime symmetry detection algorithms make them good candidates for parallel evaluation on the 8 and 16 core processors that are predicted to emerge over the next 5 years. Although the speedups achieved by parallel evaluation of BDD operations have often been modest [MJH98], the weak coupling between the iterations of the main loop of the symmetry detection algorithms – the property that yields to anytime execution – also leads to weakly coupled parallel execution.

Specifically, this thesis makes original contributions to the understanding and manipulation of Boolean functions represented as ROBDDs. In terms of novelty, the thesis contributes original algorithms for classical symmetry detection, generalised symmetry detection and ROBDD approximation. All these algorithms share the unique property that they are anytime, tractable and amenable to an efficient implementation. The validity of these theoretically inspired algorithms has been extensively evaluated and

demonstrated for a variety of benchmarks across different architectures. Because the contributions made in the thesis to both symmetry detection and ROBDD approximation are themselves multifaceted, we summarise the contributions to classical symmetry detection, generalised symmetry detection and ROBDD approximation as follows.

### 7.1.1 Contributions to Classical Symmetry Detection

We present a novel anytime algorithm for first-order classical symmetry detection for Boolean functions represented as ROBDDs. Unlike previous schemes that seek to directly discover pairs of variables that are symmetric, the algorithm systematically eliminates pairs of variables which are found to be asymmetric. The remaining variable pairs are deemed symmetric. This iterative formulation leads to an incremental anytime algorithm that opens up new opportunities for optimisation, such as applying transitivity and linear time asymmetry sieves. The overall complexity of this new algorithm is $O(n^3 + n|G| + |G|^3)$ where $n$ is the number of variables and $|G|$ the number of nodes in the ROBDD, hence the algorithm is asymptotically superior to the state-of-the-art technique [Mis03]. This improvement is significant when computing the symmetries of a Boolean function represented by a large ROBDD.

### 7.1.2 Contributions to Generalised Symmetry Detection

We show how the anytime approach may be refined so as to detect all $T_1, \ldots, T_{12}$-generalised symmetry types with only a marginal increase in complexity. To take advantage of the iterative nature of the generalised algorithm, new transitivity result have been derived which take the form, that if ${T_p}^{x_i,x_j}(f)$ and ${T_q}^{x_j,x_k}(f)$ hold then ${T_r}^{x_i,x_j}(f)$ holds where $T_p, T_q$ and $T_r$ denote one of the 12 generalised symmetry types. To the best of our knowledge, only a few of these transitivity results were previously known [TM97]. The resulting generalised symmetry detection algorithm resides in $O(n^3 + n^2|G| + |G|^3)$ and performs favourably against the state-of-the-art method [ZMBCJ06] on a wide range of benchmarks.

### 7.1.3 Contributions to ROBDD Approximation

We present a novel technique for widening Boolean functions represented as ROBDDs which is based upon prime implicants. The widening can be used to widen for both time and space. Widening for time amounts to bounding the number of times an iterative fixed-point calculation is reapplied whilst widening for space reduces to deriving a *dense* approximation of an ROBDD. The method can be used to widen from above or below. The widening is incremental, generating approximations whose accuracy is monotonically increasing as progressively longer implicants are considered. As a consequence, the widening is naturally anytime and can be tuned to achieve the desired tradeoff between

precision and density. In contrast to previous approximation techniques, the widening is independent of variable ordering, that is, the approximations obtained are invariant with respect to the underlying variable ordering. Furthermore, the widening can be realised by combining prime enumeration methods with cardinality constraints.

This widening naturally leads to the idea of randomly selecting prime implicants from which to construct the approximation, this leads to a new class of widening algorithm which can achieve accurate approximation without excessive ROBDD manipulation. Experimental work demonstrates the viability of this approach and the original prime enumeration method.

## 7.2 Directions for Future Work

### 7.2.1 ROBDD Approximation

The success of the randomised approach for discovering prime implicants suggests that there exists some underlying probabilistic or combinatoric reason as to why small implicants are likely to be selected first. This warrants an analysis of the likelihood of finding a prime implicant of minimal size in a given number of random selections. An answer to this question would provide insight into the approximability of the `SHORTEST IMPLICANT[ROBDD]` problem.

### 7.2.2 Program Analysis

This thesis on ROBDDs was motivated by the need to apply ROBDDs in program analysis. During the course of this research an example implementation was constructed. It was the intractability of the analysis and the unpredictability of existing approximation methods which led to the work reported in this dissertation. Furthermore, symmetry detection methods, which can be used to inform the variable reordering process, were also found to have poor scalability. This in turn led to the desire to build an anytime symmetry detection method with which to improve variable reordering. This inspired the second main theme of the thesis. Now the approximation and symmetry detection problems have been addressed, it would be interesting to return to the analysis problem itself. Quite apart from solving the original analysis problem, this would constitute a useful strength test for the techniques devised in this thesis.

The analysis which motivated this thesis attempted to encode primitive operations, such as binary arithmetic, with Boolean functions in order to locate security vulnerabilities in programs written in C. Rather unusually, this encoding permits syntactically different, but semantically equivalent, program fragments to be detected (at least for straight line code). This problem is at the heart of compiler optimisations such as common sub-expression elimination, and code compression techniques [BFG+03]. In fact, an ROBDD encoding enables two procedures to be checked for equivalence up to

variable reordering by reducing this problem to the so-called unknown input correspondence problem [CM93a]. As far as we are aware, this connection is not known within the compiler community and since symmetry detection is key to solving the unknown input correspondence problem, the techniques developed in this thesis could well prove useful in compiling.

### 7.2.3 Complexity Analysis

Although Mishchenko argues that his algorithm is cubic in the number of nodes, as has been pointed out in this thesis, the algorithm actually requires a cubic number of set operations each of which has variable complexity. It would be interesting therefore to derive a tight upper bound in terms of the total number of atomic operations. Allied with this, it would also be instructive to compute a lower bound on the complexity of the classical symmetry detection problem for ROBDDs. Deriving a lower bound complexity on the classical co-factor symmetry computation problem would be insightful since it would not only allow us to gauge the efficiency of currently known algorithms, but also motivate the discovery of faster algorithms, should the complexity bound be lower than that already achieved.

# Bibliography


[ABA95] P. Agrawal, D. Bhattacharya, and V. D. Agrawal. Test Generation for Path Delay Faults Using Binary Decision Diagrams. *IEEE Transactions on Computers*, 44(3):434–447, 1995.

[ADG91] P. Ashar, S. Devadas, and A. Ghosh. Boolean Satisfiability and Equivalence Checking using General Binary Decision Diagrams. In *International Conference on Computer Design*, pages 259–264. IEEE Computer Society, 1991.

[Ake78] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):506–516, 1978.

[AMSS98] T. Armstrong, K. Marriott, P. Schachte, and H. Søndergaard. Two classes of Boolean Functions for Dependency Analysis. *Science of Computer Programming*, 31(1):3–45, 1998.

[Apt03] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[ARMS02] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving Difficult SAT Instances in the Presence of Symmetry. In *Design Automation Conference*, pages 731–736. ACM Press, 2002.

[AS72] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1972.

[BB03] C. Bartzis and T. Bultan. Construction of Efficient BDDs for Bounded Arithmetic Constraints. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, pages 294–408. Springer, 2003.

[BCM+92] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic Model Checking: $10^{20}$ States and Beyond. *Information and Compututation*, 98(2):142–170, 1992.

[BEN08] Lgsynth93 Benchmark Set, 2008. http://www.bdd-portal.org/benchmarks/.

[BFG+03] A. Beszédes, R. Ferenc, T. Gyimóthy, A. Dolenc, and K. Karsisto. Survey of Code-Size Reduction Methods. *ACM Computing Surveys*, 35(3):223–267, 2003.

[BGHZ04] R. Bagnara, R. Gori, P. M. Hill, and E. Zaffanella. Finite-Tree Analysis for Constraint Logic-Based Languages. *Information and Computation*, 193(2):84–116, 2004.

[BHRZ05] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise Widening Operators for Convex Polyhedra. *Science of Computer Programming*, 58(1-2):28–56, 2005.

[BRB90] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *Design Automation Conference*, pages 40–45. ACM Press, 1990.

[Bry86] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Bry91] R. E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Application to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, 1991.

[Bry92] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[Bry96] R. E. Bryant. Bit-Level Analysis of an SRT Divider Circuit. In *Design Automation Conference*, pages 661–665. ACM Press, 1996.

[BS99] R. Bagnara and P. Schachte. Factorizing Equivalent Variable Pairs in ROBDD-Based Implementations of *Pos*. In *Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1999.

[BSM05] J. T. Butler, T. Sasao, and M. Matsuura. Average Path Length of Binary Decision Diagrams. *IEEE Transactions on Computers*, 54(9):1041–1053, 2005.

[BW96] B. Bollig and I. Wegener. Improving the Variable Ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Symposium on Principles of Programming Languages*, pages 238–252. ACM Press, 1977.

[CC79] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Symposium on Principles of Programming Languages*, pages 269–282. ACM Press, 1979.

[CC92a] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.

[CC92b] P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, editors, *International Symposium on Programming Language Implementation and Logic Programming*, Lecture Notes in Computer Science, pages 269–295. Springer, 1992.

[CCF+05] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. The ASTRÉE Analyser. In S. Sagiv, editor, *European Symposium on Programming*, volume 3444 of *Lecture Notes in Computer Science*, pages 21–30. Springer, 2005.

[CGP00] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[CH09] Y. Crama and P. L. Hammer. Boolean Functions: Theory, Algorithms and Applications, 2009. http://www.rogp.hec.ulg.ac.be/Crama/Publications/BookPage.html.

[CJ01] M. Chrzanowska-Jeske. Generalized Symmetric Variables. In *International Conference on Electronics, Circuits, and Systems*, volume 3, pages 1147–1150. IEEE Computer Society, 2001.

[CM78] A. K. Chandra and G. Markowsky. On The Number of Prime Implicants. *Discrete Mathematics*, 24(1):7–11, 1978.

[CM92a] O. Coudert and J. C. Madre. Implicit and Incremental Computation of Primes and Essential Primes of Boolean Functions. In *Design Automation Conference*, pages 36–39. IEEE Computer Society, 1992.

[CM92b] O. Coudert and J. C. Madre. A New Method to Compute Prime and Essential Prime Implicants of Boolean Functions. In *MIT Conference on Advanced Research in VLSI and Parallel Systems*. IEEE Computer Society, 1992.

[CM93a] D. I. Cheng and M. Marek Sadowska. Verifying Equivalence of Functions with Unknown Input Correspondence. In *Design Automation Conference*, pages 272–277. ACM Press, 1993.

[CM93b] O. Coudert and J. C. Madre. A New Graph Based Prime Computation Technique. In *Logic Synthesis and Optimization*. Kluwer Academic Publishers, 1993.

[Cod99] M. Codish. Worst-Case Groundness Analysis using Positive Boolean Functions. *Journal of Logic Programming*, 41(1):125–128, 1999.

[Coo71] S. A. Cook. On the Complexity of Theorem-proving Procedures. In M. A. Harrison, R. B. Banerji, and J. D. Ullman, editors, *Symposium on the Theory of Computing*, pages 151–158. ACM Press, 1971.

[Cou] O. Coudert. Two Open Questions On ROBDDs and Prime Implicants. http://www.informatik.uni-trier.de/Design_and_Test/abstract30.html (November, 2005).

[CSS99] M. Codish, H. Søndergaard, and P. J. Stuckey. Sharing and Groundness Dependencies in Logic Programs. *ACM Transactions on Programming Languages and Systems*, 21(5):948–976, 1999.

[Dav82] M. Davis. Hilbert's Tenth Problem is Unsolvable. *American Mathematical Montly*, 80:233–269, 1982.

[DB88] T. Dean and M. Boddy. An Analysis of Time-Dependent Planning. In H. Shrobe, editor, *Proceedings of the 7th National Conference on Artificial Intelligence*, pages 49–54. AAAI Press, 1988.

[DLSM04] P. T. Darga, M. H. Liffiton, K. A. Sakallah, and I. L. Markov. Exploiting Structure in Symmetry Detection for CNF. In S. Malik, L. Fix, and A. B. Kahng, editors, *Design Automation Conference*, pages 530–534. ACM Press, 2004.

[EGD04] R. Ebendt, W. Günther, and R. Drechsler. Minimization of the Expected Path Length in BDDs Based on Local Changes. In M. Imai, editor, *Asia and South Pacific Design Automation Conference*, pages 865–870. IEEE Press, 2004.

[EH78] C. R. Edward and S. L. Hurst. A Digital Synthesis Procedure Under Function Symmetries and Mapping Methods. *IEEE Transactions on Computers*, C-27(11):985–997, 1978.

[Fec97] C. Fecht. *Abstrakte Interpretation logischer Programme: Theorie, Implementierung, Generierung.* PhD thesis, Universität des Saarlandes, 1997.

[FG06] J. Flum and M. Grohe. *Parameterized Complexity.* Springer, 2006.

[FGH+05] L. Fix, O. Grumberg, A. Heyman, T. Heyman, and A. Schuster. Verifying Very Large Industrial Circuits Using 100 Processes and Beyond. In D. Peled and Yih-K. Tsay, editors, *Automated Technology for Verification and Analysis*, volume 3707 of *Lecture Notes in Computer Science*, pages 11–25. Springer, 2005.

[FL97] W. Feng and J. W. S. Liu. Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines. *IEEE Transactions on Software Engineering*, 23(2):93–106, 1997.

[Flo62] R. W. Floyd. Algorithm 97: Shortest Path. *Communications of the ACM*, 5(6):345, 1962.

[FS90] S. J. Friedman and K. J. Supowit. Finding the Optimal Variable Ordering for Binary Decision Diagrams. *IEEE Transactions on Computers*, 39(5):710–713, 1990.

[FYBSV93] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli. Dynamic Variable Reoredering for BDD Minimization. In *Design Automation Conference*, pages 130–135. ACM Press, 1993.

[GCA95] J. M. Gallone, F. Charpillet, and F. Alexandre. Anytime Scheduling with Neural Networks. In *Emerging Technologies and Factory Automation*, volume 1, pages 509–520. IEEE Press, 1995.

[GH06] L. Gonnord and N. Halbwachs. Combining Widening and Acceleration in Linear Relation Analysis. In K. Yi, editor, *The Static Analysis Symposium (SAS)*, volume 4134 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2006.

[GHB05] J. P. Gallagher, K. S. Henriksen, and G. Banda. Techniques for Scaling Up Analyses Based on Pre-interpretations. In *International Conference on Logic Programming*, volume 3668 of *Lecture Notes in Computer Science*, pages 280–296. Springer, 2005.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability a Guide to the Theory of NP-Completeness.* Freeman and Company, 1979.

[GK08] S. Genaim and A. King. Inferring Non-Suspension Conditions for Logic Programs with Dynamic Scheduling. *ACM Transactions on Computational Logic*, 2008. to appear.

[GS05] R. Grosu and S. A. Smolka. Monte Carlo Model Checking. In N. Halbwachs and L. D. Zuck, editors, *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 271–286. Springer, 2005.

[HACK00] A. Heaton, M. Abo-Zaed, M. Codish, and A. King. A Simple Polynomial Groundness Analysis for Logic Programs. *Journal of Logic Programming*, 45:143–156, 2000.

[Hay95] K. Hayase. On Relationship between Boolean Functions and Their Prime Implicants in OBDD Size. In *6th BDD Workshop*, 1995. http://www-imai.is.s.u-tokyo.ac.jp/publist/BDD.html.

[HLM00] L. Heinrich-Litan and P. Molitor. Least Upper Bounds for the Size of OBDDs Using Symmetry Properties. *IEEE Transactions on Computers*, 49:360–368, 2000.

[HLS04] P. Hawkins, V. Lagoon, and P. J. Stuckey. Set Bounds and (Split) Set Domain Propagation Using ROBDDs. In *Proceedings of the 17th Australian Joint Conference in Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 706–717. Springer, 2004.

[HLS05] P. Hawkins, V. Lagoon, and P. J. Stuckey. Solving Set Constraint Satisfaction Problems using ROBDDs. *Journal of Artificial Intelligence Research*, 24:109–156, 2005.

[HNR68] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths in Graphs. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[HS96] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[IK83] T. Ibaraki and N. Katoh. On-line Computation of Transitive Closure of Graphs. *Information Processing Letters*, 16:95–97, 1983.

[ISO99] International Organization for Standardization. ISO/IEC 9899:1999 - programming Languages – C, 1999.

[JV00] D. Jackson and M. Vaziri. Finding Bugs with a Constraint Solver. In *International Symposium on Software Testing and Analysis*, pages 14–25. ACM Press, 2000.

[Kar72] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[KD91] B. G. Kim and D. L. Dietmeyer. Multilevel Logic Synthesis of Symmetric Switching Functions. *IEEE Transactions on Computer-Aided Design*, 10(4):436–446, 1991.

[KEL71] D. J. Kleitman, M. Edelberg, and D. Lubell. Maximal Sized Antichains in Partial Orders. *Discrete Mathematics*, 1(1):47–53, 1971.

[KFM06] W. W. Kywe, D. Fujiwara, and K. Murakami. Scheduling of Image Processing Using Anytime Algorithm for Real-time System. In *International Conference on Pattern Recognition*, pages 1095–1098. IEEE Computer Society, 2006.

[KK06] N. Kettle and A. King. An Anytime Symmetry Detection Algorithm for ROBDDs. In F. Hirose, editor, *Asia and South Pacific Design Automation Conference*, pages 243–248. IEEE Press, 2006.

[KKS06] N. Kettle, A. King, and T. Strzemecki. Widening ROBDDs with Prime Implicants. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2006.

[KS00a] V. N. Kravets and K. A. Sakallah. Constructive Library-Aware Synthesis using Symmetries. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 208–215. IEEE Computer Society, 2000.

[KS00b] V. N. Kravets and K. A. Sakallah. Generalized Symmetries in Boolean Functions. In E. Sentovich, editor, *International Conference on Computer-Aided Design*, pages 526–532. IEEE Computer Society, 2000.

[KST93] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[LM91] X. Lai and J. L. Massey. A Proposal for a New Block Encryption Standard. In I. Damgård, editor, *Advances in Cryptology - Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, Lecture Notes in Computer Science, pages 389–404. Springer, 1991.

[LNO06] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras. SMT Techniques for Fast Predicate Abstraction. In T. Ball and R. B. Jones, editors, *Computer Aided Verification*, volume 4144 of *Lecture Notes in Computer Science*, pages 424–437. Springer, 2006.

[LS02] V. Lagoon and P. J. Stuckey. Precise Pair-Sharing Analysis of Logic Programs. In *Proceedings of the International Conference on Principles and Practice of Declarative Programming*, pages 99–108. ACM Press, 2002.

[LS04] V. Lagoon and P. J. Stuckey. Set Domain Propagation Using ROBDDs. In *International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2004.

[LSP92] Y. T. Lai, S. Sastry, and M. Pedram. Boolean Matching Using Binary Decision Diagrams with Applications to Logic Synthesis and Verification. In *International Conference on Computer-Aided Design*, pages 452–458. IEEE Computer Society, 1992.

[Mau98] L. Mauborgne. Abstract Interpretation Using Typed Decision Graphs. *Science of Computer Programming*, 31(1):91–112, 1998.

[McK81] B. D. McKay. Practical Graph Isomorphism. *Congeressus Numerantium*, 30:45–87, 1981.

[MD90] F. Mailhot and G. De Micheli. Technology Mapping Using Boolean Matching and Don't Care Sets. In *Design Automation Conference*, pages 212–216. ACM Press, 1990.

[Meh02] N. Mehta. Remote Buffer Overflow Vulnerability in Sun RPC, July 2002. http://xforce.iss.net/xforce/alerts/id/advise126.

[MEL06] Mailenable ltd. Mailenable v2.33, 2006. http://www.mailenable.com/.

[Min93] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Design Automation Conference*, pages 272–277. ACM Press, 1993.

[Min96] S. Minato. Graph-based Representations of Discrete Functions. In T. Sasao and M. Fujita, editors, *Representations of Discrete Functions*. Kluwer Academic Publishers, 1996. Chapter 1.

[Mis03] A. Mishchenko. Fast Computation of Symmetries in Boolean Functions. *IEEE Transactions on Computer-Aided Design*, 22(11):1588–1593, 2003.

[Mis08] A. Mishchenko. Extra Library of DD Procedures, 2008. http://www.ee.pdx.edu/∼alanmi/research/extra.htm.

[MJH98] K. Milvang-Jensen and A. J. Hu. BDDNOW: A Parallel BDD Package. In G. Gopalakrishnan and P. J. Windley, editors, *International Conference on Formal Methods in Computer-Aided Design*, volume 1522 of *Lecture Notes in Computer Science*, pages 501–507. Springer, 1998.

[MM93] J. Mohnke and S. Malik. Permutation and Phase Independent Boolean Comparison. *INTEGRATION, The VLSI Journal*, 16:109–129, 1993.

[MMM02] J. Mohnke, P. Molitor, and S. Malik. Limits of Using Signatures for Permutation Independent Boolean Comparison. *Formal Methods in System Design*, 21(2):167–191, 2002.

[MMW93] D. Möller, J. Mohnke, and M. Weber. Detection of Symmetry of Boolean functions Represented by ROBDDs. In M. R. Lightner and J. A. G. Jess, editors, *International Conference on Computer-Aided Design*, pages 680–684. IEEE Computer Society, 1993.

[MOS98] V. M. Manquinho, A. L. Oliveira, and J. P.Marques Silva. Models and Algorithms for Computing Minimum-Size Prime Implicants. In *Proceedings of the International Workshop on Boolean Problems*, 1998.

[MP64] F. Mileto and G. Putzolu. Average Values of Quantities Appearing in Boolean Function Minimization. *IEEE Transactions on Electronic Computers*, EC-13(2):87–92, 1964.

[MS97] C. Meinel and A. Slobodova. Speeding up Variable Reordering of OBDDs. In *International Conference on Computer Design*, pages 338–343. IEEE Computer Society, 1997.

[MWBSV88] S. Malik, A. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In *International Conference on Computer-Aided Design*, pages 6–9. IEEE Computer Society, 1988.

[NB86] R. Nair and D. Brand. Construction of Optimal DCVS Trees. Technical Report RC-11863, IBM Research Report, Thomas J. Watson Research Center, 1986.

[NMSB03] S. Nagayama, A. Mishchenko, T. Sasao, and J. Butler. Minimization of Average Path Length in BDDs by Variable Reordering. In D. Marculescu, editor, *International Workshop on Logic Synthesis*, pages 207–213, 2003.

[NNH05] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, 2005.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putman–Logemann–Loveland Procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[Piz96] C. Pizzuti. Computing Prime Implicants by Integer Programming. In *International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, 1996.

[Plo70] G. Plotkin. A Note on Inductive Generalisation. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[Pol75] J. M. Pollard. A Monte Carlo Method for Factorization. *BIT Numerical Mathematics*, 15(3):331–334, 1975.

[Pos41] E. L. Post. The Two-Valued Iterative Systems of Mathemtical Logic. *Annals of Mathematical Studies*, 5:1–122, 1941.

[Pre88] B. R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++*. Wiley, 1988.

[PSP94] S. Panda, F. Somenzi, and B. F. Plessier. Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. In J. A. G. Jess and R. L. Rudell, editors, *International Conference on Computer-Aided Design*, pages 628–631. IEEE Computer Society, 1994.

[Qui52] W. V. Quine. The Problem of Simplifying Truth Functions. *American Mathematical Monthly*, (52):521–531, 1952.

[Rab80] M. O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[RB04] S. S. Ramani and S. Bhanja. Any-time Probabilistic Switching Model using Bayesian Networks. In Rajiv V. Joshi, K. Choi, V. Tiwari, and K. Roy, editors, *International Symposium on Low Power Electronics and Design*, pages 86–89. ACM Press, 2004.

[Ric53] H. G. Rice. Classes of Recursively Enumerable Sets and Their Decision Problems. *Transactions of the American Mathematical Society*, 74(2):358–366, 1953.

[RMSS98] K. Ravi, K. L. McMillan, T. R. Shiple, and F. Somenzi. Approximation and Decomposition of Binary Decision Diagrams. In *Proceedings of the Design Automation Conference*, pages 445–450. IEEE Computer Society, 1998.

[RS95] K. Ravi and F. Somenzi. High-density Reachability Analysis. In R. L. Rudell, editor, *International Conference on Computer-Aided Design*, pages 154–158. IEEE Computer Society, 1995.

[Rud93] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In M. R. Lightner and J. A. G. Jess, editors, *International Conference on Computer-Aided Design*, pages 42–47. IEEE Computer Society, 1993.

[Sas99] T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishing, 1999.

[Sha38] C. E. Shannon. A Symbolic Analysis of Relay and Switching Circuits. *AIEE Transactions*, 57:713–723, 1938.

[Sha48] C. E. Shannon. The Synthesis of Two-Terminal Switching Function. *Bell System Technical Journal*, 1948.

[Shi96] T. R. Shiple. *Formal Analysis of Synchronous Circuits*. PhD thesis, University of California at Berkeley, Electronics Research Laboratory, 1996.

[Sie02] Detlef Sieling. The Nonapproximability of OBDD Minimization. *Information and Computation*, 172(2):103–138, 2002.

[SMMD99] C. Scholl, D. Möller, P. Molitor, and R. Drechsler. BDD Minimization Using Symmetries. *IEEE Transactions on Computer-Aided Design*, 18(2):81–100, 1999.

[Som05] F. Somenzi. CUDD Package, Release 2.4.1, 2005. http://vlsi.colorado.edu/~fabio/CUDD/.

[SR95] R. C. Sekar and I. V. Ramakrishnan. Fast Strictness Analysis Based on Demand Propagation. *ACM Transactions on Programming Languages and Systems*, 17(6):896–937, 1995.

[SS06] P. Schachte and H. Søndergaard. Closure Operators for ROBDDs. In *International Conference on Verification, Model Checking and Abstract Interpretation*, volume 3855 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2006.

[Str92] T. Strzemecki. Polynomial-Time Algorithms for Generation of Prime Implicants. *Journal of Complexity*, 8(1):37–63, 1992.

[SW93] D. Sieling and I. Wegener. Reduction of OBDDs in Linear Time. *Information Processing Letters*, 48:139–144, 1993.

[TM97] C. C. Tsai and M. Marek-Sadowska. Boolean Functions Classification via Fixed Polarity Reed-Muller Forms. *IEEE Transactions on Computers*, 46(2):173–186, 1997.

[Tse68] G. Tseitin. On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic*, 2:115–125, 1968.

[TW69] C. B. Tompkins and W. L. Wilson. *Elementary Numerical Analysis*. Prentice-Hall, 1969.

[Uma99] C. Umans. On the Complexity and Inapproximability of Shortest Implicant Problems. In *International Colloqium on Automata, Languages and Programming*, volume 1644 of *Lecture Notes in Computer Science*, pages 687–696. Springer, 1999.

[Uma01] C. Umans. The Minimum Equivalent DNF Problem and Shortest Implicants. *Journal of Computer and System Sciences*, 63(4):597–611, 2001.

[War62] S. Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. Society for Industrial Mathematics, 2000.

[Wei] E. W. Weisstein. Tautology. http://mathworld.wolfram.com/Tautology.html.

[WKSV03] G. Wang, A. Kuehlmann, and A. L. Sangiovanni-Vincentelli. Structural Detection of Symmetries in Boolean Functions. In *International Conference on Computer Design*, pages 498–503. IEEE Computer Society, 2003.

[WL04] J. Whaley and M. S. Lam. Cloning-Based Context-Sensitive Pointer Alias Analysis Using Binary Decision Diagrams. In W. Pugh and C. Chambers, editors, *Programming Language Design and Implementation*, pages 131–144. ACM Press, 2004.

[ZCJMB04] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch. Generalized Symmetries in Boolean Functions: Fast Computation and Application to Boolean Matching. In D. Marculescu, editor, *International Workshop on Logic Synthesis*, pages 424–430, 2004.

[ZCJMB05] J. S. Zhang, M. Chrzanowska-Jeske, A. Mishchenko, and J. R. Burch. Detecting Support-Reducing Bound Sets using Two-Cofactor Symmetries. In T-A. Tang, editor, *Asia and South Pacific Design Automation Conference*, pages 266–271. ACM Press, 2005.

[Zil96] S. Zilberstein. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, 17:73–83, 1996.

[ZMBCJ06] J. S. Zhang, A. Mishchenko, R. Brayton, and M. Chrzanowska-Jeske. Symmetry Detection for Large Boolean Functions using Circuit Representation, Simulation, and Satisfiability. In E. Sentovich, editor, *Design Automation Conference*, pages 510–515. ACM Press, 2006.

# Appendix A

# Proof of Generalised Symmetry Relations

**Proposition A.1.** Table 7 of Chapter 4 summarises a collection of results that state implicational relationships between various generalised symmetries. For example, if ${T_3}^{x_i,x_j}(f)$ and ${T_4}^{x_j,x_k}(f)$ hold for some ROBDD $f$ then ${T_3}^{x_i,x_k}(f)$ also holds.

*Proof.*

- Suppose ${T_1}^{x,y}$ and ${T_3}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow =0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_4}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus

$f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_5}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_{11}}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_3}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus

$f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_4}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_1}^{x,y}$ and ${T_5}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_{11}}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_7}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} f|_{x\leftarrow 1,z\leftarrow 1}$ and $symm6^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_3}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_4}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_5}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_{11}}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus

$f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_3}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_4}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_5}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_{11}}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_8}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.
- Suppose ${T_8}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.
- Suppose ${T_3}^{x,y}$ and ${T_3}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.
- Suppose ${T_3}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.
- Suppose ${T_3}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.
- Suppose ${T_3}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.
- Suppose ${T_9}^{x,y}$ and ${T_9}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 0,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_9}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_3}^{x,z}$ holds.

- Suppose ${T_9}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,z\leftarrow 1}$ and ${T_9}^{x,z}$ holds.

- Suppose ${T_4}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_4}^{x,z}$ holds.

- Suppose ${T_4}^{x,y}$ and ${T_{10}}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_{10}}^{x,y}$ and ${T_4}^{y,z}$ hold. Thus $f|_{x\leftarrow 1,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 1,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{10}}^{x,z}$ holds.

- Suppose ${T_5}^{x,y}$ and ${T_5}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_5}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$,

thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_5}^{x,z}$ holds.

- Suppose ${T_5}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_5}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0}$, hence $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_{11}}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 0} = \neg f|_{y\leftarrow 1,z\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$. Hence $f|_{x\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,z\leftarrow 0}$ and ${T_{11}}^{x,z}$ holds.

- Suppose ${T_{11}}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_{11}}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_6}^{x,y}$ and ${T_6}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_6}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1}$, hence $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$,

thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_6}^{x,z}$ holds.

- Suppose ${T_{12}}^{x,y}$ and ${T_{12}}^{y,z}$ hold. Thus $f|_{x\leftarrow 0,y\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 0} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 0}$ and $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Also $f|_{y\leftarrow 0,z\leftarrow 1} = \neg f|_{y\leftarrow 1,z\leftarrow 1}$, thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1}$ and $f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Thus $f|_{x\leftarrow 0,y\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 0,z\leftarrow 1}$ and moreover $f|_{x\leftarrow 0,y\leftarrow 1,z\leftarrow 1} = \neg f|_{x\leftarrow 1,y\leftarrow 1,z\leftarrow 1}$. Hence $f|_{x\leftarrow 0,z\leftarrow 1} = \neg f|_{x\leftarrow 1,z\leftarrow 1}$ and ${T_{12}}^{x,z}$ holds.

- Suppose ${T_2}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_2}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_2}^{y,x}$ hold, whence ${T_2}^{z,x}$ and ${T_2}^{x,z}$.

- Suppose ${T_3}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_5}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_5}^{y,x}$ hold, whence ${T_5}^{z,x}$ and ${T_3}^{x,z}$.

- Suppose ${T_3}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_5}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_5}^{y,x}$ hold, whence ${T_{11}}^{z,x}$ and ${T_9}^{x,z}$.

- Suppose ${T_3}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_5}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_5}^{y,x}$ hold, whence ${T_5}^{z,x}$ and ${T_3}^{x,z}$.

- Suppose ${T_3}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_5}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_5}^{y,x}$ hold, whence ${T_{11}}^{z,x}$ and ${T_9}^{x,z}$.

- Suppose ${T_9}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_{11}}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_{11}}^{y,x}$ hold, whence ${T_{11}}^{z,x}$ and ${T_9}^{x,z}$.

- Suppose ${T_9}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_{11}}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_{11}}^{y,x}$ hold, whence ${T_5}^{z,x}$ and ${T_3}^{x,z}$.

- Suppose ${T_9}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_{11}}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_{11}}^{y,x}$ hold, whence ${T_{11}}^{z,x}$ and ${T_9}^{x,z}$.

- Suppose ${T_9}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_{11}}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_{11}}^{y,x}$ hold, whence ${T_5}^{z,x}$ and ${T_3}^{x,z}$.

- Suppose ${T_9}^{x,y}$ and ${T_3}^{y,z}$ hold. Therefore ${T_{11}}^{y,x}$ and ${T_5}^{z,y}$ hold, hence ${T_5}^{z,y}$ and ${T_{11}}^{y,x}$ hold, whence ${T_5}^{z,x}$ and ${T_3}^{x,z}$.

- Suppose ${T_4}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.

- Suppose ${T_4}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_4}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_4}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_4}^{x,y}$ and ${T_3}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_5}^{z,y}$ hold, hence ${T_5}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_4}^{x,y}$ and ${T_9}^{y,z}$ hold. Therefore ${T_6}^{y,x}$ and ${T_{11}}^{z,y}$ hold, hence ${T_{11}}^{z,y}$ and ${T_6}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_3}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_5}^{z,y}$ hold, hence ${T_5}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_9}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_{11}}^{z,y}$ hold, hence ${T_{11}}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_{12}}^{z,x}$ and ${T_{10}}^{x,z}$.
- Suppose ${T_{10}}^{x,y}$ and ${T_4}^{y,z}$ hold. Therefore ${T_{12}}^{y,x}$ and ${T_6}^{z,y}$ hold, hence ${T_6}^{z,y}$ and ${T_{12}}^{y,x}$ hold, whence ${T_6}^{z,x}$ and ${T_4}^{x,z}$.
- Suppose ${T_5}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_3}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_3}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_5}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_3}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_3}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_5}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_3}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_3}^{y,x}$ hold, whence ${T_4}^{z,x}$ and ${T_6}^{x,z}$.
- Suppose ${T_5}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_3}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_3}^{y,x}$ hold, whence ${T_4}^{z,x}$ and ${T_6}^{x,z}$.

- Suppose ${T_{11}}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_9}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_9}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_{11}}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_9}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_9}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_{11}}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_9}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_9}^{y,x}$ hold, whence ${T_{10}}^{z,x}$ and ${T_{12}}^{x,z}$.
- Suppose ${T_{11}}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_9}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_9}^{y,x}$ hold, whence ${T_{10}}^{z,x}$ and ${T_{12}}^{x,z}$.
- Suppose ${T_{11}}^{x,y}$ and ${T_5}^{y,z}$ hold. Therefore ${T_9}^{y,x}$ and ${T_3}^{z,y}$ hold, hence ${T_3}^{z,y}$ and ${T_9}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_4}^{z,x}$ and ${T_6}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_4}^{z,x}$ and ${T_6}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_5}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_3}^{z,y}$ hold, hence ${T_3}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_6}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Therefore ${T_4}^{y,x}$ and ${T_9}^{z,y}$ hold, hence ${T_9}^{z,y}$ and ${T_4}^{y,x}$ hold, whence ${T_3}^{z,x}$ and ${T_5}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_1}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_1}^{z,y}$ hold, hence ${T_1}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_{10}}^{z,x}$ and ${T_{12}}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_7}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_7}^{z,y}$ hold, hence ${T_7}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_{10}}^{z,x}$ and ${T_{12}}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_2}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_2}^{z,y}$ hold, hence ${T_2}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_8}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_8}^{z,y}$ hold, hence ${T_8}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_5}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_3}^{z,y}$ hold, hence ${T_3}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.

- Suppose ${T_{12}}^{x,y}$ and ${T_{11}}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_9}^{z,y}$ hold, hence ${T_9}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_9}^{z,x}$ and ${T_{11}}^{x,z}$.
- Suppose ${T_{12}}^{x,y}$ and ${T_6}^{y,z}$ hold. Therefore ${T_{10}}^{y,x}$ and ${T_4}^{z,y}$ hold, hence ${T_4}^{z,y}$ and ${T_{10}}^{y,x}$ hold, whence ${T_{10}}^{z,x}$ and ${T_{12}}^{x,z}$.

□