Decision Diagrams for Ternary Expressions [1]

by

Peter Mayo

An essay
presented to the University of Waterloo
in fulfilment of the
essay requirement for the degree of
Masters of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1993

[1]This work was supported by the Information Technology Research Centre of Ontario and the Natural Sciences and Engineering Research Council of Canada under grant OGP0000871.

Abstract

We present an extension of Binary Decision Diagrams used to treat ternary expressions. The use of these diagrams in a computer program called "T.E.R.M." (Ternary Expression Reduction Machine) is presented. T.E.R.M.is a program designed to allow a human to direct the reduction of a ternary expression to a desired form. It also allows the user to automatically compare two arbitrary expressions for equality. It attempts to do this in a reasonably efficient manner. The worst case of these algorithms is probably doomed to exhibit exponential growth in the size of the input, since 3-SAT is directly solvable by the program and 3-SAT is NP-complete. However, some useful cases are computationally feasible.

Acknowledgements

I would like to thank Professor C-J. Seger of the University of British Columbia for suggesting the topic as an interesting project. I would also like to thank Professor J. A. Brzozowski for many things, first for his patience and support throughout my studies, and secondly for his editorial input, and constructive comments.

Personally, I would like to thank Professor Brzozowski, and the many others, who told me to "keep going, and get it done" when continuing the essay seemed fruitless. These include, but are not limited to, Rob Kroeger, Sue Guckenberger, Wendy Rush, Jo Ebergen, and Josie Vitello.

The MAVERIC research group has also earned my acknowledgements for regular discussions of motivational and related topics. I would like to thank the Computer Graphics Lab for providing the late night editing environment, and useful input in the coding and presentation of the T.E.R.M. program, especially Richard Stroobosscher for his comments on the code itself.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the analysis of digital circuits it is natural to consider every signal as having one of two values, and this is part of the strength of digital circuit techniques. These values change in time, and while it is often convenient to treat these changes as instantaneous, in physical implementations these changes can take significant periods of time. During that interval the value may be interpreted inconsistently by the various components. Extending the set of possible values to include a third, ambiguous, value is a natural and useful way to model this situation.

There are other situations in which a value distinct from the logic values is useful, such as the case of a "floating" line, which is neither being driven positive, nor being pulled to ground. These arise naturally in the modelling of MOS circuits, though they may be undesirable in the final result. Rules for handling the effects of such control lines are then required. Such rules produce fewer simplifications than the standard binary models, and working with these situations can be tedious. In particular, some routine analysis and verification methods yield fairly complex expressions. Many of the questions involving such expressions can be viewed as determining their equivalence to certain simple design goals.

The manipulation of ternary expressions is often subtle; people accustomed to binary simplification may easily be lulled by the similarity of the two algebras and perform an incorrect transformation. Also such expressions tend to convey relatively little meaning for the number of symbols they contain. For both of these reasons the aid of computers would be of benefit.

The problem we wish to address within this paper is to provide a reasonably efficient tool to manipulate ternary expressions in a natural way. In doing so we would like to be able to compare expressions, and aid a user in finding and expressing equivalent forms in a timely manner.

1.2 History

The use of ternary (three-valued) systems to model relay circuits was discussed in 1949 by M. Goto [12]. The third value he introduced has been used to model undetermined behaviour, unknown initial conditions, and transient behaviours. In 1964 Yoeli and Rinon produced a paper on the application of ternary algebra to the study of static hazards [24]. In 1965 Eichelberger developed a method for detection of hazards (intermediate unexpected results) and critical races (where the final stable state may depend on the timing of the components). This method has been subjected to study, and its relationship to other models analyzed [8, 23]. In more recent works ternary-valued models have been used for more accurate analysis of MOS circuits [2, 7, 9, 21], and may lead to more methods in the years to come. Applications of computer manipulation techniques to these problems has been successful [10, 14, 13].

Eichelberger's method and further references can be found in Brzozowski and Seger's 1990 paper[5, 7]. The work on race and hazard detection seemed to regain interest with renewal of work on asynchronous circuits in the mid-80s and beyond. In synchronous systems, the simple solution of slowing the clock rate will reduce the problems associated with hazards, and race conditions can be localized and managed. A hazard in an asynchronous network is usually a more serious problem.

There has also been a some progress in the area of the underlying algebra of the ternary set used in these methods. In particular, Mukaidono [17, 18, 19] has investigated many interesting properties of the structure. Of special interest is the relationship between ternary extensions, ternary expressions, and monotonic functions given in his 1986 work[19]. These works have led to the integration of a more complete understanding of ternary objects, beyond their immediate application. We have chosen a formalism almost identical to that of Brzozowski and Seger[7], Chapter 2, as it was closely tied to the intended use of the work.

The field of graphical manipulation of Boolean functions has received much interest since the publication of Bryant's 1986 paper[3]. This work has been implemented by others [1, 16]. The general finite-valued function graph has been covered briefly in Srinivasan et al. [22]. Their implementation uses a transformation to the binary case, using many binary variables to encode each of the variables in larger finite spaces. The difference between a re-implementation and a translation to a simple base case is sometimes quite marked, so we have chosen to implement the desired case rather than possibly inefficiently reduce it to the first work (Binary Decision Diagrams).

There is an excellent summary of recent work in areas related to Binary Decision Diagrams and their extensions in Bryant's 1992 survey paper[4]. This is recommended as a starting place for a deeper background and understanding of the principles involved.

1.3 Conventions

In the following we will use $\mathbb{N}$ to represent the set of positive integers $1, 2, 3, \ldots$. A vector whose elements are a_1 through a_n will be denoted by $(a_1, \ldots, a_n)$. Function evaluation will be denoted by appending parenthesized arguments to the name; thus $f(1)$ represents the evaluation of f at the domain value 1 and is the unique element of the range associated with 1. Functions of vector arguments will generally have the second enclosing set of parentheses removed, except where the meaning would become unclear. We will introduce further notation later.

Chapter 2

Definitions

2.1 Ternary Algebra

For our definitions regarding the basics of ternary algebra we will follow the work of [7] almost literally. It is included here as a base for what we will discuss later. A *ternary algebra* is an algebraic system $\langle \mathcal{A}, +, \cdot, ^{-}, 0, \Phi, 1 \rangle$, where $\mathcal{A}$ is a set, $+$ and $\cdot$ are binary operations on $\mathcal{A}$, $^{-}$ is a unary operation on $\mathcal{A}$, and 0, 1 and Φ are three distinct elements of $\mathcal{A}$ such that all the laws of Table 2.1 are satisfied.

We shall often be concerned with operations in the simplest ternary algebra, $\mathbf{T} = \langle \{0, \Phi, 1\}, +, \cdot, ^{-}, 0, \Phi, 1 \rangle$. Here the axioms of Table 2.1 completely define the operations $+$, $\cdot$, and $^{-}$, as shown in table 2.2.

2.2 Ternary Functions

Again following [7] we consider ternary-valued functions of many ternary variables. A *ternary function* of n variables is any function $\mathbf{f}$ from $\{0, \Phi, 1\}^n$ into $\{0, \Phi, 1\}$, for $n > 0$. To each n-tuple $\mathbf{a} = (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \{0, \Phi, 1\}^n$, the function $\mathbf{f}$ assigns a unique value $\mathbf{f}(\mathbf{a}) \in \{0, \Phi, 1\}$.

Table 2.1: Laws of Ternary Algebra

T1	$\mathbf{a}+\mathbf{a} = \mathbf{a}$	T1′	$\mathbf{a}\cdot\mathbf{a} = \mathbf{a}$
T2	$\mathbf{a}+\mathbf{b} = \mathbf{b}+\mathbf{a}$	T2′	$\mathbf{a}\cdot\mathbf{b} = \mathbf{b}\cdot\mathbf{a}$
T3	$\mathbf{a}+(\mathbf{b}+\mathbf{c}) = (\mathbf{a}+\mathbf{b})+\mathbf{c}$	T3′	$\mathbf{a}\cdot(\mathbf{b}\cdot\mathbf{c}) = (\mathbf{a}\cdot\mathbf{b})\cdot\mathbf{c}$
T4	$\mathbf{a}+(\mathbf{a}\cdot\mathbf{b}) = \mathbf{a}$	T4′	$\mathbf{a}\cdot(\mathbf{a}+\mathbf{b}) = \mathbf{a}$
T5	$\mathbf{a}+0 = \mathbf{a}$	T5′	$\mathbf{a}\cdot 1 = \mathbf{a}$
T6	$\mathbf{a}+1 = 1$	T6′	$\mathbf{a}\cdot 0 = 0$
T7	$\overline{(\overline{\mathbf{a}})} = \mathbf{a}$		
T8	$\mathbf{a}+(\mathbf{b}\cdot\mathbf{c}) = (\mathbf{a}+\mathbf{b})\cdot(\mathbf{a}+\mathbf{c})$	T8′	$\mathbf{a}\cdot(\mathbf{b}+\mathbf{c}) = (\mathbf{a}\cdot\mathbf{b})+(\mathbf{a}\cdot\mathbf{c})$
T9	$\overline{(\mathbf{a}+\mathbf{b})} = \overline{\mathbf{a}}\cdot\overline{\mathbf{b}}$	T9′	$\overline{(\mathbf{a}\cdot\mathbf{b})} = \overline{\mathbf{a}}+\overline{\mathbf{b}}$
T10	$\mathbf{a}+\overline{\mathbf{a}}+\Phi = \mathbf{a}+\overline{\mathbf{a}}$	T10′	$\mathbf{a}\cdot\overline{\mathbf{a}}\cdot\Phi = \mathbf{a}\cdot\overline{\mathbf{a}}$
T11	$\Phi = \overline{\Phi}$		
	For all $\mathbf{a}, \mathbf{b}$, and $\mathbf{c}$ in $\mathcal{A}$.		

Table 2.2: The Ternary Functions $+$, $\cdot$, and $^-$

	$\mathbf{a}_1+\mathbf{a}_2$	$\mathbf{a}_2$ = 0	Φ	1
	0	0	Φ	1
$\mathbf{a}_1$	Φ	Φ	Φ	1
	1	1	1	1

	$\mathbf{a}_1\cdot\mathbf{a}_2$	$\mathbf{a}_2$ = 0	Φ	1
	0	0	0	0
$\mathbf{a}_1$	Φ	0	Φ	Φ
	1	0	Φ	1

$\mathbf{a}$	$\overline{\mathbf{a}}$
0	1
Φ	Φ
1	0

We extend the definitions of $+$ and $\cdot$ to the space of ternary functions of n variables in the natural manner, namely

$$(\mathbf{f}+\mathbf{g})(\mathbf{a}) \stackrel{\text{def}}{=} \mathbf{f}(\mathbf{a})+\mathbf{g}(\mathbf{a}),$$

$$(\mathbf{f}\cdot\mathbf{g})(\mathbf{a}) \stackrel{\text{def}}{=} \mathbf{f}(\mathbf{a})\cdot\mathbf{g}(\mathbf{a}),$$

$$\bar{\mathbf{f}}(\mathbf{a}) \stackrel{\text{def}}{=} \overline{(\mathbf{f}(\mathbf{a}))}.$$

Note that on the right side we deal with functions in T, whereas on the left we are introducing new functions. We then introduce three special functions $\mathbf{0}$, Φ and $\mathbf{1}$ as follows: For all $\mathbf{a} \in \{0, \Phi, 1\}^n$,

$$\mathbf{0}(\mathbf{a}) \stackrel{\text{def}}{=} 0 \qquad \Phi(\mathbf{a}) \stackrel{\text{def}}{=} \Phi \qquad \mathbf{1}(\mathbf{a}) \stackrel{\text{def}}{=} 1.$$

This allows us to quote the following theorem [7]

Theorem 1 *Let $\mathbf{T}_n$ be the set of all ternary functions of n variables. Then $\mathbf{T}_n$ is a ternary algebra under the operations defined above with $\mathbf{0}$, Φ, and $\mathbf{1}$, acting as 0, Φ, and 1.*

Proof: We need to verify that the operations on functions as given above satisfy the laws of Table 2.1. This is quite straightforward, and we leave the details to the interested reader. □

2.3 Ternary Expressions

We now turn to the definition of the objects we wish to manipulate. Again we turn to [7] for the appropriate basics. Ternary expressions will be shown to have a many to one correspondence with the ternary functions, and it is the determination of the equivalence classes of this correspondence that is the purpose of this work.

We will wish to use a set of variables, and for clarity we require that these symbols be distinct from the constants, 0, Φ, and 1. The set of variables may be arbitrarily large, but we do not gain anything useful from allowing the set to be infinite; hence we restrict the set to be finite.

Let 0, Φ, 1, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be distinct symbols. A *ternary expression* over $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is defined inductively:

1. 0, Φ, 1, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are ternary expressions.
2. If E and F are ternary expressions, then so are $(E{+}F)$, $(E{\cdot}F)$, and $\overline{E}$.
3. Nothing else is a ternary expression unless its being so follows from a finite number of applications of Rules 1 and 2.

Now, let $\mathcal{F}_n$ denote the set of all ternary expressions over n variables. Define a mapping $|\ |\colon \mathcal{F}_n \rightarrow \mathbf{T}_n$ as follows:

1. The expressions 0, Φ, 1, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are mapped to the functions $\mathbf{0}$, $\mathbf{\Phi}$, $\mathbf{1}$, $|\mathbf{x}_1|, \ldots, |\mathbf{x}_n|$, respectively, where $|\mathbf{x}_i|$ is defined by

$$|\mathbf{x}_i|(\mathbf{a}_1, \ldots, \mathbf{a}_n) \stackrel{\text{def}}{=} \mathbf{a}_i \text{ for all } (\mathbf{a}_1, \ldots, \mathbf{a}_n) \in \{0, \Phi, 1\}^n.$$

2.

$$\begin{aligned} |(E{+}F)| &\stackrel{\text{def}}{=} (|E|{+}|F|) \\ |(E{\cdot}F)| &\stackrel{\text{def}}{=} (|E|{\cdot}|F|) \\ |\overline{E}| &\stackrel{\text{def}}{=} \overline{|E|}. \end{aligned}$$

The mapping $|\ |$ assigns to each expression $E \in \mathcal{F}_n$ a unique ternary function $|E| \in \mathbf{T}_n$. Again, it should be emphasized that there is only one ternary function corresponding to a given ternary expression, but a given ternary function can be represented by an infinite number of distinct ternary expressions. Note that the mapping $|\ |$ is not onto, that is there are ternary functions which have no ternary expression representations. For example, the single variable function that is 1 on Φ, and 0 otherwise, (the characteristic function of Φ), has no corresponding ternary expression. See also [7], section 2.3.

2.4 The Uncertainty Partial Order

This order represents the amount of certainty in a ternary value, where Φ represents an "uncertain" or "unsure" value. This name makes more sense when considering the application to vectors or functions, but all can be based on the simple case of $\{0,\ \Phi,\ 1\}$. This order has been considered by many authors when characterizing ternary functions. [7, 8, 18, 17, 20]

We define the partial order $\sqsubseteq$ on $\{0,\ \Phi,\ 1\}$by $0 \sqsubseteq 0$, $1 \sqsubseteq 1$, $\Phi \sqsubseteq \Phi$, $0 \sqsubseteq \Phi$, $1 \sqsubseteq \Phi$, and no other relations hold. We then extend $\sqsubseteq$ to $\mathbf{a}$, $\mathbf{b}$ in $\{0, \Phi, 1\}^n$ by

$$\mathbf{a} \sqsubseteq \mathbf{b} \text{ if and only if } (\mathbf{a}_i \sqsubseteq \mathbf{b}_i \text{ for all } i),$$

and, moreover, we extend it to all $\mathbf{f}$, $\mathbf{g}$ in $\mathbf{T}_n$ by

$$\mathbf{f} \sqsubseteq \mathbf{g} \text{ if and only if } \mathbf{f}(\mathbf{a}) \sqsubseteq \mathbf{g}(\mathbf{a}) \text{ for all } \mathbf{a} \text{ in } \{0, \Phi, 1\}^n.$$

Notice then that

$$\begin{aligned} x \sqsubseteq 1 &\Rightarrow x = 1, \\ x \sqsubseteq 0 &\Rightarrow x = 0, \\ (x_1, \ldots, x_n) &\sqsubseteq (x_1, \ldots, x_{i-1}, \Phi, x_{i+1}, \ldots, x_n). \end{aligned}$$

A function $\mathbf{f}$ in $\mathbf{T}_n$ is defined to be *monotonic* if, for all $\mathbf{a}$, $\mathbf{a}' \in \{0, \Phi, 1\}^n$,

$$\mathbf{a} \sqsubseteq \mathbf{a}' \text{ implies } \mathbf{f}(\mathbf{a}) \sqsubseteq \mathbf{f}(\mathbf{a}').$$

We now notice an interesting property of $\sqsubseteq$, namely that for all $\mathbf{a}$, $\mathbf{a}'$, $\mathbf{b}$, $\mathbf{b}'$ in $\{0,\ \Phi,\ 1\}$, if $\mathbf{a} \sqsubseteq \mathbf{a}'$ and $\mathbf{b} \sqsubseteq \mathbf{b}'$ then

$$\begin{aligned} \mathbf{a}+\mathbf{b} &\sqsubseteq \mathbf{a}'+\mathbf{b}', \\ \mathbf{a}\cdot\mathbf{b} &\sqsubseteq \mathbf{a}'\cdot\mathbf{b}', \\ \text{and } \overline{\mathbf{a}} &\sqsubseteq \overline{\mathbf{a}'}. \end{aligned}$$

This comes from examining the definitions of $+$, $\cdot$, and $^{-}$. By an exercise in quantifier manipulation we see that this implies that for $\mathbf{f}$, $\mathbf{f}'$, $\mathbf{g}$, $\mathbf{g}'$ in $\mathbf{T}_n$, if $\mathbf{f} \sqsubseteq \mathbf{f}'$ and $\mathbf{g} \sqsubseteq \mathbf{g}'$ then

$$\begin{aligned} \mathbf{f}+\mathbf{g} &\sqsubseteq \mathbf{f}'+\mathbf{g}', \\ \mathbf{f}\cdot\mathbf{g} &\sqsubseteq \mathbf{f}'\cdot\mathbf{g}', \\ \text{and } \overline{\mathbf{f}} &\sqsubseteq \overline{\mathbf{f}'}. \end{aligned}$$

The above property then leads us to propose the following theorem.

Theorem 2 *If* $\mathbf{f}$ *and* $\mathbf{g}$ *are monotonic ternary functions then so are* $\mathbf{f}+\mathbf{g}$*,* $\mathbf{f}\cdot\mathbf{g}$*, and* $\overline{\mathbf{f}}$*.*

Proof: Assume $\mathbf{f}$, $\mathbf{g}$ are monotonic ternary functions. Assume $\mathbf{a}$,$\mathbf{a}'$ in $\{0, \Phi, 1\}^n$ and $\mathbf{a} \sqsubseteq \mathbf{a}'$.

Since $\mathbf{f}$ and $\mathbf{g}$ are monotonic, we have

$$\mathbf{f}(\mathbf{a}) \sqsubseteq \mathbf{f}(\mathbf{a}') \text{ and } \mathbf{g}(\mathbf{a}) \sqsubseteq \mathbf{g}(\mathbf{a}').$$

Now by the observed property

$$\mathbf{f}(\mathbf{a})+\mathbf{g}(\mathbf{a}) \sqsubseteq \mathbf{f}(\mathbf{a}')+\mathbf{g}(\mathbf{a}').$$

But

$$\begin{aligned} (\mathbf{f}+\mathbf{g})(\mathbf{a}) &= \mathbf{f}(\mathbf{a})+\mathbf{g}(\mathbf{a}), \\ \text{and } (\mathbf{f}+\mathbf{g})(\mathbf{a}') &= \mathbf{f}(\mathbf{a}')+\mathbf{g}(\mathbf{a}'); \end{aligned}$$

thus $(\mathbf{f}+\mathbf{g})(\mathbf{a}) \sqsubseteq (\mathbf{f}+\mathbf{g})(\mathbf{a}')$.

Similarly, if $\mathbf{a}$, $\mathbf{a}'$ are in $\{0, \Phi, 1\}^n$ and $\mathbf{a} \sqsubseteq \mathbf{a}'$, then

$$(\mathbf{f}\cdot\mathbf{g})(\mathbf{a}) = \mathbf{f}(\mathbf{a})\cdot\mathbf{g}(\mathbf{a}) \sqsubseteq \mathbf{f}(\mathbf{a}')\cdot\mathbf{g}(\mathbf{a}') = (\mathbf{f}\cdot\mathbf{g})(\mathbf{a}')$$

$$\text{and } \overline{\mathbf{f}}(\mathbf{a}) = \overline{\mathbf{f}(\mathbf{a})} \sqsubseteq \overline{\mathbf{f}(\mathbf{a}')} = \overline{\mathbf{f}}(\mathbf{a}') :$$

thus $\overline{\mathbf{f}}$, $\mathbf{f}\cdot\mathbf{g}$, and $\mathbf{f}+\mathbf{g}$ are monotonic functions. □

This result for the operators leads to the following theorem for the representations of ternary expressions.

Theorem 3 *If* F *is a ternary expression then* $|F|$ *is monotonic.*

Proof: We only sketch the proof here; for further details see [17]. Suppose F is a ternary expression formed by Rule 1. If F is one of 0, 1, Φ, then for all $\mathbf{a}$, $\mathbf{a}'$ in $\{0, \Phi, 1\}^n$ $|F|(\mathbf{a}) \sqsubseteq |F|(\mathbf{a}')$ since $0 \sqsubseteq 0$, $1 \sqsubseteq 1$, and $\Phi \sqsubseteq \Phi$. If F is $\mathbf{x}_i$ then suppose $\mathbf{a}$, $\mathbf{a}'$ in $\{0, \Phi, 1\}^n$ and $\mathbf{a} \sqsubseteq \mathbf{a}'$. In particular, $\mathbf{a}_i \sqsubseteq \mathbf{a}'_i$; thus $|F|(\mathbf{a}) = \mathbf{a}_i \sqsubseteq \mathbf{a}'_i = |F|(\mathbf{a}')$.

Now we make the inductive hypothesis that, for all $n < N$, if F is a ternary expression formed by n applications of Rule 2 and an arbitrary (finite) number of applications of Rule 1 then $|F|$ is monotonic. Now for the inductive step we assume F is derived from N applications of Rule 2.

Case 1: $F = (F' + E')$. Now F' and E' are derived using fewer than N applications of Rule 2, and thus $|F'|$ and $|E'|$ are monotonic. Hence $|F| = |F'| + |E'|$ is monotonic by the previous theorem.

Case 2: $F = (F' \cdot E')$. Now F' and E' are derived using fewer than N applications of Rule 2, and thus $|F'|$ and $|E'|$ are monotonic. Hence $|F| = |F'| \cdot |E'|$ is monotonic by the previous theorem.

Case 3: $F = (\overline{E})$. Now E is derived using fewer than N applications of Rule 2, and thus $|E|$ is monotonic. Hence $|F| = \overline{|E|}$ is monotonic by the previous theorem.

Thus by the principle of mathematical induction, $|F|$ is monotonic if F is derived by any finite number of applications of Rules 1 and 2. By Rule 3 no other objects are ternary expressions; thus, for any ternary expression F, $|F|$ is monotonic. $\square$

While this theorem should be of no surprise to those accustomed to recursive definitions and function spaces, the surprising result is that the converse is true, namely that every monotonic ternary function has a ternary expression representation. We do not attempt to prove the converse here as it is not required to discuss our work; the interested reader should see Mukaidono's work[17], or chapter 2 of Brzozowski and Seger's work[7]. We will show later that every monotonic ternary function has a ternary expression representation by providing an algorithm which produces one; however, this involves the construction of an intermediate data structure whose complexity is much greater than is required to prove this result.

2.5 Expression Trees

Expression trees are graphical representations of expressions. They are well-known and thus their theoretical properties are assumed as background knowledge. A quick recursive definition of our use of them follows.

1. $0, \Phi, 1, x_1, \ldots, x_n$ are expression trees.

2. if E and F are expression trees then $+(E,F)$, $\cdot(E,F)$, and $^{-}(E)$ are expression trees.

3. nothing else is an expression tree.

There is a natural one-to-one correspondence between expression trees and expressions given by:

$$\begin{array}{rcl} +(E,F) & \leftrightarrow & (E{+}F) \\ \cdot(E,F) & \leftrightarrow & (E{+}F) \\ ^{-}(E) & \leftrightarrow & (\overline{E}) \\ 0 \leftrightarrow 0, & \ldots, & x_n \leftrightarrow x_n \end{array}$$

We extend the definition of $|E|$, where E is an expression, to expression trees by saying that $|T|$ is the function represented by the expression corresponding to T. Alternatively, a similar recursive definition may be given for $|T|$.

The corresponding graphical structure uses the expression trees as vertices. The edge relation is derived from calling E and F the children of $+(E,F)$, $\cdot(E,F)$, $^{-}(E)$. This, with appropriate formalism, gives an expression tree the same structure as a graph-thoeoretic tree.

Two expression trees, say E and F, are *equivalent* if $|E| = |F|$, i.e. if they represent the same function. Since the function represented by any vertex in an expression tree depends only on its type and the functions represented by its children, any subtree may be replaced by an equivalent subtree without changing the function represented by the entire tree. More formally if $|E| = |E'|$ and $|F| = |F'|$ then $|+(E,F)| = |+(E',F')|$, $|\cdot(E,F)| = |\cdot(E',F')|$, $|^{-}(E)| = |^{-}(E')|$.

The above equivalence provides the basis for most step-by-step proofs of equalities from axioms. For example, from the ternary axiom $a{+}a = a$, we conclude that for all expression trees E, $|+(E,E)| = |E| + |E| = |E|$. This allows us to replace an expression tree like $+(E,E)$ with E, or vice-versa. An algorithm for applying such manipulations will be presented in section 4.4.

2.6 Decision Diagrams

We wish to use an efficient representation of a ternary function which will allow us to easily decide the equivalence of two ternary expressions. To this end we will define ternary decision diagrams. We will later prove that they satisfy certain desirable properties.

A *Ternary Decision Diagram* can be viewed as a rooted labelled directed acyclic graph, in which each vertex is either an external vertex, whose outdegree is zero, or an internal vertex whose outdegree is three. The labels on the vertices correspond to variables. The labels on arcs correspond to possible assignments to the variables. External vertices are labelled by unique values from the set $\{0, \Phi, 1\}$. See Figure 2.1 for a simple example; it will be explained more fully later. The idea of such a graph is that it represents an efficient way to compute the result of the application of the function it represents to an arbitrary assignment to the variables. Then to evaluate a given vertex we examine the value assigned to the variable corresponding to the label on the vertex and follow the arc with the same label to the next step of the computation, stopping when we get to an external vertex and returning the value we find there.

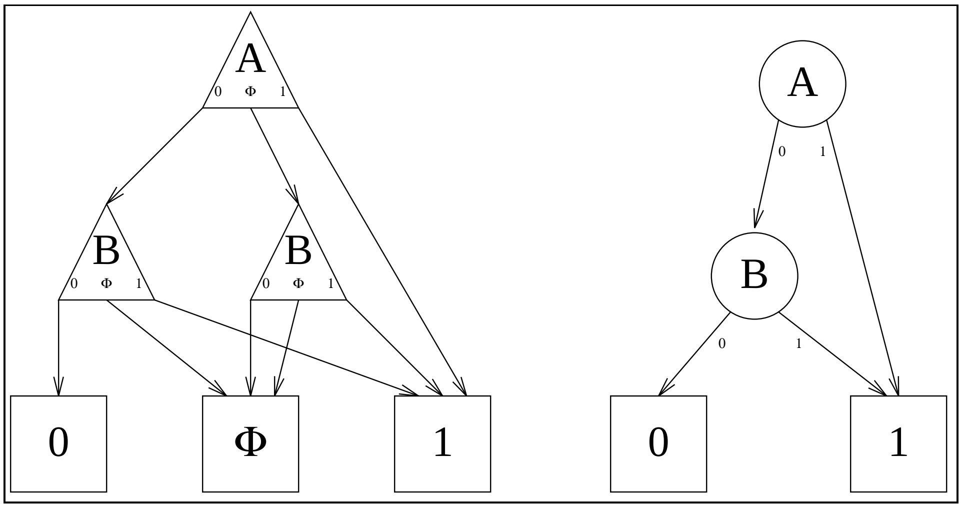


Figure 2.1: Pictorial representation of a Ternary Decision Diagram and a similar Binary Decision Diagram

A Ternary Decision Diagram turns out to be a unique representation for a function if the order in which we examine the variables is fixed and redundancy is removed. There is nothing inherently ternary about the idea, and it is no easier to define the specific instance. Consequently we will begin by defining a generic decision diagram.

A *Decision Diagram* is a six-tuple $< A, V, d, n, l, r >$ where

1. A is a finite set of "values", distinguishable from elements of $\mathbb{N}$.
2. V is a finite set of "vertices".
3. d is a function from $V \times A$ to V and represents the edges of the graph.
4. l is a function from V to $\mathbb{N} \cup A$. l represents the label on a vertex, which determines which variable, or constant, is associated with the vertex.
5. r is a distinguished member of V, called the root.
6. For every v in V, such that $l(v)$ is in $\mathbb{N}$, $l(d(v, a)) < l(v)$ or $l(d(v, a)) \in A$ for every a in A. Notice that this requirement eliminates the possibility of loops by transitivity and the strictness of the inequality.
7. If $v \in V$ and $l(v) \in A$, then there is no other y in V such that $l(y) = l(v)$. Moreover $d(v, a) = v$ for all a in A. These vertices are referred to as external vertices.
8. For v, y in V, if $l(v) = l(y)$ and $d(v, a) = d(y, a)$ for all a in A, then $v = y$. This gives us the uniqueness which will be of great use later.

We now define a *Ternary Decision Diagram* to be a Decision Diagram with $A = \{0, \Phi, 1\}$.

We introduce a common operation of Ternary Decision Diagrams, namely following the successor function of the root. We define, for all $a \in A$,

$$< A, V, d, l, r >:a \stackrel{\text{def}}{=} < A, V, d, l, d(r, a) > .$$

We also define

$$\text{ord}(< A, V, d, l, r >) \stackrel{\text{def}}{=} l(r).$$

We now turn our attention to defining a functional interpretation of a Decision Diagram. More formally we define a mapping $|\ |_{A,n}$ from the set of all Decision Diagrams $D = < A, V, d, l, r >$ with $l(r) \leq n$ to the functions from A^n to A by:

For all $(a_1, \ldots, a_n)$ in A^n

$$|D|_{A,n}(a_1, \ldots, a_n) = \begin{cases} l(r) & \text{if } l(r) \text{ is in } A, \\ |D{:}a_{l(r)}|_{A,l(r)-1}(a_1, \ldots, a_{l(r)-1}) & \text{otherwise.} \end{cases}$$

We must show that the mapping $|\ |_{A,n}$ is defined for all the elements in its domain. We accomplish this by noting that the value of the label of the root decreases at each step, and that it is bounded below by zero. After at most n iterations, we must be able to apply the first rule. In this way we have determined a unique element of A for each element of A^n, and thus have achieved our goal of associating a function with each Decision Diagram.

An observation can be made that will simplify further arguments, for reference we will call this the *extension property.*

Theorem 4 *Let $D = < A, V, d, l, r >$ be a decision diagram. If $l(r) \leq n < m$ then $|D|_{A,m}$, $|D|_{A,n}$ are defined and, for all $(a_1, \ldots, a_m)$ in A^m, $|D|_{A,m}(a_1, \ldots, a_m) = |D|_{A,n}(a_1, \ldots, a_n)$.*

Proof: By the definition of $|D|_{A,n}$, since both $m, n \geq l(r)$ then $|D|_{A,n}$ and $|D|_{A,m}$ are defined.

If $l(r)$ is in A then both $|D|_{A,n}$ and $|D|_{A,m}$ are the constant $l(r)$, otherwise both are equal to $|D{:}a_{l(r)}|_{A,l(r)-1}(a_1, \ldots, a_{l(r)-1})$, and hence equal to each other. □

This observation leads us to abuse the formalism slightly to clarify our notation. When dealing with functions we will omit the specification of the size of the domain, except where it is important. Without loss of generality the domain can be chosen to be A^n for n sufficiently large.

Since we plan to deal with expressions as well as Decision Diagrams, we will often refer to the variable that corresponds to the label of the root of a Decision

Diagram. Thus we make the following definition for notation purposes:

$$\mathrm{var}(< A, V, d, l, r >) \stackrel{\mathrm{def}}{=} x_{l(r)} \text{for } l(r) \notin A.$$

We can combine this with the previous results to give a more compact definition for $|D|$ where D is a Decision Diagram:

$$|D|(x_1, \ldots, x_n) = \begin{cases} |D{:}\mathrm{var}(T)|(x_1, \ldots, x_n), & \text{if } \mathrm{ord}(D) \in I\!N \\ \mathrm{ord}(D), & \text{if } \mathrm{ord}(D) \in A. \end{cases}$$

We notice now that we may in fact apply the recursive definition more often than necessary for some diagrams. In particular, a vertex whose successor does not depend on the input variable (i.e. is the same for all three values) is not informative. This leads us to make the following definition, similar to Bryant's notion of reduced Binary Decision Diagrams[3].

A vertex v in V of a Decision Diagram $< A, V, d, l, r >$ is said to be *redundant* if $l(v)$ is in $I\!N$ and there is a y in V such that, for all a in A, $d(v, a) = y$. A Decision Diagram is said to be redundant if there is a redundant vertex v in its vertex set V. An *irredundant* Decision Diagram is any Decision Diagram which is not redundant.

We will sketch the transformation from a redundant Decision Diagram to a non-redundant Decision Diagram; but this construction serves only as motivation, and so will not be given in detail. This is called reduction in Bryant's work[3].

A redundant Decision Diagram $< A, V, d, l, r >$ may be transformed by choosing a redundant vertex v in V and defining $d_1(y, a) = d(v, a)$ if $d(y, a) = v$, otherwise $d_1(y, a) = d(y, a)$; and $r_1 = d(v, a)$ if $r = v$, otherwise $r_1 = r$. Then we see that

$$| < A, V, d, l, r > |_A \;=\; | < A, V, d_1, l, r_1 > |_A.$$

This may be repeated until there are no more redundant vertices. Since a single vertex is eliminated at every step, and there are a finite number of vertices to start with, eventually we can eliminate no more vertices.

We now show the major use of Decision Diagrams, namely that they are a functional representation. This means that only identical (or trivially different) Decision Diagrams represent the same function. We eliminate the trivial differences by making the seemingly strong requirement that the two diagrams be defined over the same sets of vertices and values using the same labelling and successor functions. These requirements actually serve to simplify the use and definition of such objects, and are not an issue in practice.

Theorem 5 *Let $D_1 =< A, V, d, l, r_1 >$, $D_2 =< A, V, d, l, r_2 >$, be two irredundant Decision Diagrams. If $|D_1|_{A,n}$, and $|D_2|_{A,n}$ are both defined for some integer n, then $|D_1|_{A,n} = |D_2|_{A,n}$ if and only if $r_1 = r_2$.*

Proof:

Let $D_1 =< A, V, d, l, r_1 >$, and $D_2 =< A, V, d, l, r_2 >$. Clearly if $r_1 = r_2$ then $D_1 = D_2$; hence $|D_1|_{A,n} = |D_2|_{A,n}$ for all n.

Let n be the smallest n such that there exist Decision Diagrams D_1, and D_2, as above, and $|D_1|_{A,n} = |D_2|_{A,n}$; but $r_1 \neq r_2$.

If $l(r_1)$ and $l(r_2)$ are both in A, then

$$l(r_1) = |D_1|_{A,n}(x_1, \ldots, x_n) = |D_2|_{A,n}(x_1, \ldots, x_n) = l(r_2).$$

Hence by property 7 of Decision Diagrams, $r_1 = r_2$, which contradicts the choice of D_1 and D_2.

If $l(r_1) = l(r_2)$ is in $I\!N$ then by property 8 of Decision Diagrams there is an a in A such that $d(r_1, a) \neq d(r_2, a)$. Now by applying the definition of $|\ |_{A,n}$ we get: for all $(a_1, \ldots, a_{l(r_1)-1})$ in $A^{l(r_1)-1}$,

$$\begin{aligned} |D_1{:}a|_{A,l(r_1)-1}(a_1, \ldots, a_{l(r_1)-1}) &= |D_1|_{A,n}(a_1, \ldots, a_{n-1}, a), \\ &= |D_2|_{A,n}(a_1, \ldots, a_{n-1}, a), \\ &= |< A, V, d, l, d(r_2, a) >|_{A,l(r_2)-1}(a_1, \ldots, a_{l(r_2)-1}). \end{aligned}$$

But we know $l(r_1) - 1 < n$ and $l(r_1) - 1 = l(r_2) - 1$, so

$$|< A, V, d, l, d(r_2, a) >|_{A,l(r_2)-1} = |< A, V, d, l, d(r_1 a) >|_{A,l(r_2)-1},$$

and n was not minimal.

If $l(r_1)$ is in $I\!N$ and $l(r_2) < l(r_1)$ or $l(r_2)$ is in A then we observe that $n = l(r_1)$ by n's minimality and the extension property. By the non-redundancy of D_1 there is an a in A such that $d(r_1, a) \neq r_2$. Then as above

$$
\begin{aligned}
&| < A, V, d, l, d(r_1, a) > |_{A,l(r_1)-1}(a_1, \ldots, a_{l(r_1)-1}) \\
&\qquad\qquad = \quad |D_1|_{A,n}(a_1, \ldots, a_{n-1}, a), \\
&\qquad\qquad = \quad |D_2|_{A,n}(a_1, \ldots, a_{n-1}, a), \\
&\qquad\qquad = \quad | < A, V, d, l, r_2 > |_{A,l(r_1)-1}(a_1, \ldots, a_{l(r_1)-1}),
\end{aligned}
$$

and again n is not minimal.

Hence no such n can exist, and we can now compare the function represented by the Decision Diagram by comparing the root.

□

The previous theorem is stated for Decision Diagrams whose components other than the root are equal. This means that the vertices, edges, label function, and set of values are all the same. In our use of these diagrams, this is easily guaranteed as the vertices are selected without duplication from a single set of vertices, and a vertex determines its own successors. It should be no great surprise that the set of values must be the same in order to state theorems about the equality of their evaluations. If they were different then the result of function evaluation would, *a priori*, result in different values.

2.7 Ternary Decision Diagrams

Recall that a Ternary Decision Diagram is a Decision Diagram whose set of values (A) happens to be $\{0, \Phi, 1\}$. The values 0 and 1 in $\{0, \Phi, 1\}$ must be considered distinct from the values 0 and 1 in $I\!N$. We will usually discuss the values in $\{0, \Phi, 1\}$ and thus we will make ourselves explicit when we refer to 0 and 1 in $I\!N$.

When we are dealing with Ternary Decision Diagrams we will generally subsume the details of sets and functions of the definition so a Ternary Decision Diagram will be assumed to have a sufficiently large vertex set, whose members are characterized by their label and possible successors. In our discussions there will be one global set of vertices, one global successor function, and one global label function for all of the Ternary Decision Diagrams. By choosing this approach we can characterise a Ternary Decision Diagram by its root, as was suggested by Theorem 5. The following operators on Ternary Decision Diagrams will clarify some of our future definitions.

$$\mathrm{root}(< \{0, \Phi, 1\}, V, d, l, r >) = r$$

$$\mathrm{tdd}(r) =< \{0, \Phi, 1\}, V, d, l, r >$$

Recall also that

$$< \{0, \Phi, 1\}, V, d, l, r >:a =< \{0, \Phi, 1\}, V, d, l, d(r, a) >$$

$$\mathrm{var}(< \{0, \Phi, 1\}, V, d, l, r >) = x_{l(r)}, \text{ for } l(r) \in I\!N.$$

We will also want to be able to specify vertices independently of their actual representation. For this purpose we will define node(l,v_0,v_Φ,v_1) to be the unique vertex v in V for which $l(v) = l$, and for which $d(v, a) = v_a$ for all a in {0, Φ, 1}. If we assume that V is constructed in such a way that node(l,v_0,v_Φ,v_1) is defined for all valid l, v_0, v_Φ, v_1, then the specific choice for the elements of V is irrelevant.

We have now mathematically defined the objects we referred to as graphical representations. It seems only fitting that, as graphs have a natural pictorial representation, so do Ternary Decision Diagrams. In the pictures we will draw each vertex as a triangle with three outgoing arcs, representing the value of $d(r, 0)$, $d(r, \Phi)$, and $d(r, 1)$ respectively. A square will be used to represent the vertices with labels in {0, Φ, 1}. No outgoing arcs are required, so they are omitted. See Figure 2.2 for examples.

Theorem 6 *If* $\mathbf{T}$ *is a Ternary Decision Diagram with ord(*$\mathbf{T}$*)* $\in I\!N$*, and* $|\mathbf{T}|$ *is monotonic, then* $|\mathbf{T}{:}a|$ *is monotonic for all* $a \in \{0, \Phi, 1\}$.

Proof: Let $\mathbf{T}$ be a Ternary Decision Diagram, as prescribed. Let $a \in \{0, \Phi, 1\}$, let $(x_1, \ldots, x_n), (y_1, \ldots, y_n) \in \{0, \Phi, 1\}^n$, and $(x_1, \ldots, x_n) \sqsubseteq (y_1, \ldots, y_n)$. This means $x_i \sqsubseteq y_i$ for all i. Since $a \sqsubseteq a$ we know

$$(x_1, \ldots, x_{\mathrm{ord}(\mathbf{T})-1}, a) \sqsubseteq (y_1, \ldots, y_{\mathrm{ord}(\mathbf{T})-1}, a).$$

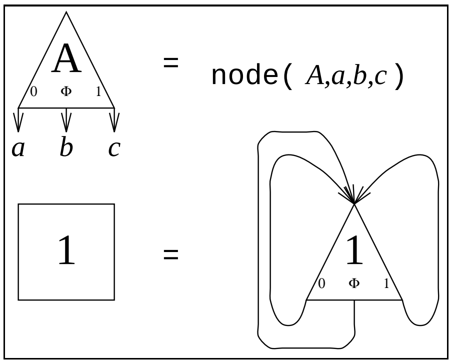


Figure 2.2: Example Ternary Decision Diagram vertices.

Thus, by monotonicity of $|\mathbf{T}|$,

$$|\mathbf{T}|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{T})-1}, a) \sqsubseteq |\mathbf{T}|(y_1, \ldots, y_{\mathrm{ord}(\mathbf{T})-1}, a).$$

Now applying the definition of $|\ |$ yields:

$$|\mathbf{T}{:}a|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{T})-1}) \sqsubseteq |\mathbf{T}{:}a|(y_1, \ldots, y_{\mathrm{ord}(\mathbf{T})-1}).$$

The extension property(Theorem 4) then tells us that

$$|\mathbf{T}{:}a|(x_1, \ldots, x_n) \sqsubseteq |\mathbf{T}{:}a|(y_1, \ldots, y_n).$$

Since a, $(x_1, \ldots, x_n)$, and $(y_1, \ldots, y_n)$ were arbitrary, $|\mathbf{T}{:}a|$ is monotonic for all a in $\{0, \Phi, 1\}$. $\square$

Chapter 3

Algorithms

When we extended the definitions of $+$, $\cdot$, and $\bar{\ }$ from the ternary algebra to the function space we did so in a straightforward manner. We now wish to proceed to an algorithmic description of such operations for Ternary Decision Diagrams that can be carried out in reasonable time. We will define constructions and prove that they have the mathematical properties we desire, rather than defining objects with mathematicalal properties and then showing constructions that are equivalent to them.

3.1 Expressions to Ternary Decision Diagrams

Given two Ternary Decision Diagrams $\mathbf{T}$ and $\mathbf{S}$, then we define $\mathbf{T}$+$\mathbf{S}$ by the following

if $\mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$ and $\mathrm{ord}(\mathbf{S}) \in \{0, \Phi, 1\}$

$$\mathbf{T}+\mathbf{S} = \mathrm{tdd}(\mathrm{ord}(\mathbf{S})+\mathrm{ord}(\mathbf{T}))$$

else if $\mathrm{ord}(\mathbf{S}) < \mathrm{ord}(\mathbf{T})$ or $\mathrm{ord}(\mathbf{S}) \in \{0, \Phi, 1\}$

$$\mathbf{T}+\mathbf{S} = \mathrm{tdd}(\mathrm{node}(\mathrm{ord}(\mathbf{T}),\mathbf{T}{:}0+\mathbf{S},\mathbf{T}{:}\Phi+\mathbf{S},\mathbf{T}{:}1+\mathbf{S}))$$

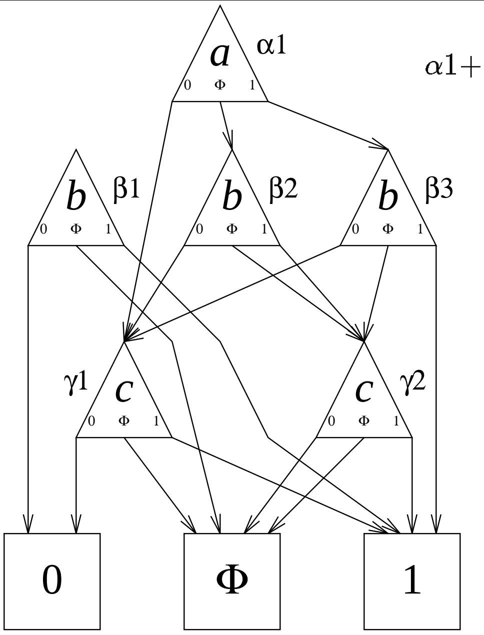
else if ord($\mathbf{S}$) $>$ ord($\mathbf{T}$) or ord($\mathbf{T}$) $\in \{0, \Phi, 1\}$

$$\mathbf{T}+\mathbf{S} = \mathrm{tdd}(\mathrm{node}(\mathrm{ord}(\mathbf{S}),\mathbf{T}+\mathbf{S}{:}0,\mathbf{T}+\mathbf{S}{:}\Phi,\mathbf{T}+\mathbf{S}{:}1))$$

else ord($\mathbf{S}$) $=$ ord($\mathbf{T}$) and both are in $I\!N$

$$\mathbf{T}+\mathbf{S} = \mathrm{tdd}(\mathrm{node}(\mathrm{ord}(\mathbf{S}),\mathbf{T}{:}0+\mathbf{S}{:}0,\mathbf{T}{:}\Phi+\mathbf{S}{:}\Phi,\mathbf{T}{:}1+\mathbf{S}{:}1))$$

An example using this definition can be found in Figure 3.1. Note that working through this example demonstrates one of the issues to be implemented. The issue is that of recognising vertices that already exist, in order to maintain Property 8 of Decision Diagrams.



$$\begin{aligned}
\alpha 1+\beta 1 &= (a, \gamma 1+\beta 1, \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, (b, \gamma 1+0, \gamma 1+\Phi, \gamma 1+1), \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, (b, \gamma 1, \gamma 1+\Phi, 1), \qquad \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, (b, \gamma 1, (c, 0+\Phi, \Phi+\Phi, 1+\Phi), 1), \\
&\qquad\qquad \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, (b, \gamma 1, (c, \Phi, \Phi, 1), 1), \quad \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, (b, \gamma 1, \gamma 2, 1), \qquad \beta 2+\beta 1, \beta 3+\beta 1) \\
&= (a, \beta 3, \qquad \beta 2+\beta 1, \beta 3+\beta 1) \\
&= \vdots \\
&= (a, \beta 3, \beta 3, \beta 3) \\
&= \beta 3
\end{aligned}$$

Figure 3.1: Pictorial representation of $a{\cdot}b{+}c{+}b$.

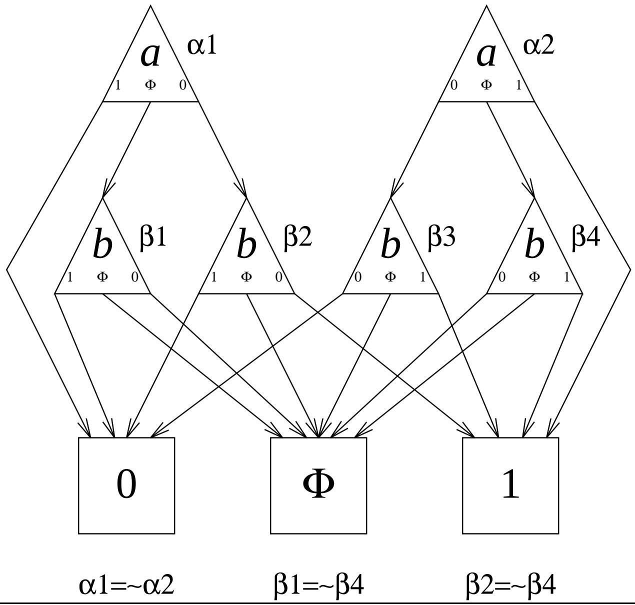
We must show that this construction terminates for it to be valid. Notice that to build a Ternary Decision Diagram by these rules, requires only Ternary Decision Diagrams of smaller, or ternary root labels; thus we need only apply the rule a finite number of times.

The definition for $\cdot$ is similar, where $+$ is replaced by $\cdot$.

The definition for $\overline{\mathbf{T}}$ can be stated more briefly, namely

$$\overline{\mathbf{T}} = \begin{cases} \mathrm{tdd}(\overline{\mathrm{ord}(\mathbf{T})}) & \text{if } \mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}, \\ \mathrm{tdd}(\mathrm{node}(\mathrm{ord}(\mathbf{T}), \overline{\mathbf{T}{:}0}, \overline{\mathbf{T}{:}\Phi}, \overline{\mathbf{T}{:}1})) & \text{otherwise.} \end{cases}$$

In order to clarify this a pictorial example for inversion is included in Figure 3.2.



$$\begin{aligned} \overline{\alpha 2} &= (a, \overline{\beta 3}, \overline{\beta 4}, \overline{1}) \\ &= (a, (b, \overline{0}, \overline{\Phi}, \overline{1}), (b, \overline{\Phi}, \overline{\Phi}, \overline{1}), 0) \\ &= (a, (b, 1, \Phi, 0), (b, \Phi, \Phi, 0), 0) \\ &= (a, \beta 2, \beta 1, 0) \\ &= \alpha 1 \end{aligned}$$

Note: the order of some successors is reversed for diagrammatic clarity

Figure 3.2: Pictorial example of inversion operation

The following theorem justifies these definitions.

Theorem 7 *For all pairs,* $(\mathbf{T}, \mathbf{S})$, *of Ternary Decision Diagrams,* $|\mathbf{T}{+}\mathbf{S}| = |\mathbf{T}|{+}|\mathbf{S}|$, $|\mathbf{T}{\cdot}\mathbf{S}| = |\mathbf{T}|{\cdot}|\mathbf{S}|$, *and* $|\overline{\mathbf{T}}| = \overline{|\mathbf{T}|}$.

Proof:

Case 1: $\mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$ and $\mathrm{ord}(\mathbf{S}) \in \{0, \Phi, 1\}$

$$|\mathbf{T}{+}\mathbf{S}| = \mathrm{ord}(\mathbf{T}){+}\mathrm{ord}(\mathbf{S}) = |\mathbf{T}|{+}|\mathbf{S}|,$$

since $|\mathrm{tdd}(\mathbf{c})| = \mathbf{c}$ for $\mathbf{c} \in \{0, \Phi, 1\}$.

Case 2: $\text{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$ or $\text{ord}(\mathbf{S}) > \text{ord}(\mathbf{T})$

Assume that a pair, $(\mathbf{T}_0, \mathbf{S}_0)$, of Ternary Decision Diagrams exists such that the theorem does not hold, i.e., $|\mathbf{T}_0{+}\mathbf{S}_0| \neq |\mathbf{T}_0|{+}|\mathbf{S}_0|$, and $\text{ord}(\mathbf{T}_0) \in \{0, \Phi, 1\}$ or $\text{ord}(\mathbf{S}_0) > \text{ord}(\mathbf{T}_0)$. Further, we may assume that $\text{ord}(\mathbf{S}_0)$ is minimal among such pairs. Now since $|\mathbf{T}_0{+}\mathbf{S}_0| \neq |\mathbf{T}_0|{+}|\mathbf{S}_0|$, choose $(c_1, \ldots, c_n)$ such that $n = \text{ord}(\mathbf{S}_0)$ and

$$|\mathbf{T}_0{+}\mathbf{S}_0|(c_1, \ldots, c_n) \neq |\mathbf{T}_0|(c_1, \ldots, c_n){+}|\mathbf{S}_0|(c_1, \ldots, c_n).$$

Such a choice must exist for the functions to be different. Applying the definition of $|\ |$ tells us that

$$|\mathbf{T}_0{+}\mathbf{S}_0|(c_1, \ldots, c_n) = |(\mathbf{T}_0{+}\mathbf{S}_0){:}c_n|(c_1, \ldots, c_n).$$

Examining the definition of $\mathbf{T}_0{+}\mathbf{S}_0$, specifically choosing the c_n sucessor, we see that

$$|(\mathbf{T}_0{+}\mathbf{S}_0){:}c_n|(c_1, \ldots, c_n) = |\mathbf{T}_0{+}(\mathbf{S}_0{:}c_n)|(c_1, \ldots, c_n).$$

Now $\text{ord}(\mathbf{S}_0{:}c_n) < \text{ord}(\mathbf{S}_0)$ or $\text{ord}(\mathbf{S}_0{:}c_n) \in \{0, \Phi, 1\}$, so by either minimality of the counterexample, or Case 1,

$$|\mathbf{T}_0{+}(\mathbf{S}_0{:}c_n)|(c_1, \ldots, c_n) = |\mathbf{T}_0|(c_1, \ldots, c_n){+}|\mathbf{S}_0{:}c_n|(c_1, \ldots, c_n).$$

We can then apply the extension property, Theorem 4, to $\mathbf{S}_0{:}c_n$, and conclude that

$$|\mathbf{T}_0{+}\mathbf{S}_0|(c_1, \ldots, c_n) = |\mathbf{T}_0|(c_1, \ldots, c_n){+}|\mathbf{S}_0|(c_1, \ldots, c_n).$$

Hence no minimal $\mathbf{S}_0$ can exist, there can be no ternary decision diagrams which violate the result, and the theorem holds in this case.

Case 3: $\text{ord}(\mathbf{S}) > \text{ord}(\mathbf{T})$ or $\text{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$

This case is completely symmetric to Case 2.

Case 4: $\text{ord}(\mathbf{S}) = \text{ord}(\mathbf{T})$ and both are in $\mathbb{N}$

As in Case 2, let $\mathbf{S}_0, \mathbf{T}_0$ be two Ternary Decision Diagrams such that

$$|\mathbf{T}_0{+}\mathbf{S}_0| \neq |\mathbf{T}_0|{+}|\mathbf{S}_0|,$$

and $\text{ord}(\mathbf{S}_0) = \text{ord}(\mathbf{T}_0)$ is minimal among such ternary decision diagrams. Now since $|\mathbf{T}_0{+}\mathbf{S}_0| \neq |\mathbf{T}_0|{+}|\mathbf{S}_0|$ choose $(c_1, \ldots, c_n)$ such that

$n = \mathrm{ord}(\mathbf{S}_0)$ and $|\mathbf{T}_0{+}\mathbf{S}_0|(c_1,\ldots,c_n) \neq |\mathbf{T}_0|(c_1,\ldots,c_n){+}|\mathbf{S}_0|(c_1,\ldots,c_n)$. Such a choice must exist for the functions to be different. Now

$$|\mathbf{T}_0{+}\mathbf{S}_0|(c_1,\ldots,c_n) = |(\mathbf{T}_0{+}\mathbf{S}_0){:}c_n|(c_1,\ldots,c_n)$$

by the definition of $|\ \ |$; the right-hand side expression is equal to

$$|\mathbf{T}_0{:}c_n{+}\mathbf{S}_0{:}c_n|(c_1,\ldots,c_n)$$

by definition of $\mathbf{T}_0{+}\mathbf{S}_0$. We know this is equal to

$$|\mathbf{T}_0{:}c_n|(c_1,\ldots,c_n){+}|\mathbf{S}_0{:}c_n|(c_1,\ldots,c_n)$$

by either minimality of the counterexample, or Case 1. Applying the extension property, Theorem 4, to $|\mathbf{T}_0{:}c_n|$ and $|\mathbf{S}_0{:}c_n|$ tells us the previous expression is equal to

$$|\mathbf{T}_0|(c_1,\ldots,c_n){+}|\mathbf{S}_0|(c_1,\ldots,c_n).$$

Hence

$$|\mathbf{T}_0{+}\mathbf{S}_0| = |\mathbf{T}_0|{+}|\mathbf{S}_0|;$$

thus no minimal $(\mathbf{S}_0,\ \mathbf{T}_0)$ can exist. Therefore there can be no ternary decision diagrams which violate the result, and the theorem holds in this case.

We have thus exhausted all possible cases for $\mathrm{ord}(\mathbf{S})$ and $\mathrm{ord}(\mathbf{T})$ and have shown that in all cases

$$|\mathbf{T}{+}\mathbf{S}| = |\mathbf{T}|{+}|\mathbf{S}|.$$

The result for $\cdot$ is symmetric: simply replace $+$ by $\cdot$. This is easily seen since the only property of $+$ used was its definition; and the definition of $\cdot$ is syntactically equivalent to $+$.

Now we must show $|\overline{\mathbf{T}}| = \overline{|\mathbf{T}|}$.

If $\mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$ then

$$\begin{aligned} |\overline{\mathbf{T}}|(a_1,\ldots,a_n) &= |\mathrm{tdd}(\overline{\mathrm{ord}(\mathbf{T})})|(a_1,\ldots,a_n) \\ &= \overline{\mathrm{ord}(\mathbf{T})}, \\ &= \overline{|\mathbf{T}|(a_1,\ldots,a_n)}. \end{aligned}$$

Hence $|\overline{\mathbf{T}}| = \overline{|\mathbf{T}|}$ if $\mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$.

Let $\mathbf{T}$ be a Ternary Decision Diagram with minimal root label in $I\!N$such that $|\overline{\mathbf{T}}| \neq \overline{|\mathbf{T}|}$. Then there is some $(c_1, \ldots, c_n)$ in A^n such that $|\overline{\mathbf{T}}|(c_1, \ldots, c_n) \neq \overline{|\mathbf{T}|}(c_1, \ldots, c_n)$. But

$$
\begin{aligned}
|\overline{\mathbf{T}}|(c_1, \ldots, c_n) = & \ |\mathrm{tdd}(\mathrm{node}(\mathrm{ord}(\mathbf{T}),\overline{\mathbf{T}{:}0},\overline{\mathbf{T}{:}\Phi},\overline{\mathbf{T}{:}1}))|(c_1, \ldots, c_n), \\
= & \ |\overline{\mathbf{T}{:}c_{\mathrm{ord}(\mathbf{T})}}|(c_1, \ldots, c_n), \\
= & \ \overline{|\mathbf{T}{:}c_{\mathrm{ord}(\mathbf{T})}|(c_1, \ldots, c_n)}, \\
= & \ \overline{|\mathbf{T}|(c_1, \ldots, c_n)}, \\
= & \ \overline{|\mathbf{T}|}(c_1, \ldots, c_n);
\end{aligned}
$$

so $|\overline{\mathbf{T}}| = \overline{|\mathbf{T}|}$. Thus no minimal counter-example exists, and the claim hoolds.

In summary we may conclude $|\mathbf{T}{+}\mathbf{S}| = |\mathbf{T}|{+}|\mathbf{S}|$, $|\mathbf{T}{\cdot}\mathbf{S}| = |\mathbf{T}|{\cdot}|\mathbf{S}|$, and $|\overline{\mathbf{T}}| = \overline{|\mathbf{T}|}$

□

We use this result to simplify the computation of $\mathbf{T}{+}\mathbf{S}$, $\mathbf{T}{\cdot}\mathbf{S}$, and $\overline{\mathbf{T}}$ in certain special cases by "shortcutting" to the desired ternary decision diagram. If $\mathrm{ord}(\mathbf{S}) = 0$, i.e., if $\mathbf{S} = \mathrm{tdd}(\mathrm{node}(0,0,0,0))$, then

$$|\mathbf{T}{+}\mathbf{S}| = |\mathbf{T}|{+}|\mathbf{S}| = |\mathbf{T}|{+}0 = |\mathbf{T}|.$$

Thus by Theorem 5 $\mathbf{T}{+}\mathbf{S} = \mathbf{T}$.

Similarly $\mathbf{T}{+}\mathrm{tdd}(\mathrm{node}(1,1,1,1)) = \mathrm{tdd}(\mathrm{node}(1,1,1,1))$, $\mathbf{T}{\cdot}\mathrm{tdd}(\mathrm{node}(1,1,1,1)) = \mathbf{T}$, and $\mathbf{T}{\cdot}\mathrm{tdd}(\mathrm{node}(0,0,0,0)) = \mathrm{tdd}(\mathrm{node}(0,0,0,0))$.

Appealing to the same two theorems, we can verify that all the axioms of ternary algebra hold for Ternary Decision Diagrams, where the operations were defined above, and the constants are simply the representations of the constant functions. More precisely if $\mathcal{W}$ is the set of all Ternary Decision Diagrams, then $\langle \mathcal{W}, +, \cdot, \bar{\ }, \mathrm{tdd}(\mathrm{node}(0,0,0,0)), \mathrm{tdd}(\mathrm{node}(\Phi,\Phi,\Phi,\Phi)), \mathrm{tdd}(\mathrm{node}(1,1,1,1)) \rangle$ is a ternary algebra.

3.2 Ternary Decision Diagrams to Expressions

When we have an object in the form of a Ternary Decision Diagram we may, from time to time, desire to produce an expression representation of this object. The first observation to be made is that not every Ternary Decision Diagram has an expression representation. We know that every expression has a unique Ternary Decision Diagram that represents the same function, and that these functions are monotonic. We may therefore assume monotonicity of the function represented by a Ternary Decision Diagram if we attempt to produce an equivalent expression.

We begin by defining two expressions for every Ternary Decision Diagram, and then we will show some useful properties of these expressions. The properties will allow us to make conclusions about the relationship between the sum of the expressions and the function represented by the Ternary Decision Diagram from which it was derived. In particular, we will determine an expression that is somehow "equivalent" to the Ternary Decision Diagram.

We define a mapping e_1 from Ternary Decision Diagrams to expressions by

$$e_1(\mathbf{T}) = \begin{cases} 0 & \text{if } \mathbf{T} \in \{0, \Phi\}, \\ 1 & \text{if } \mathbf{T} = 1, \\ (e_1(\mathbf{T}{:}\Phi) + ((\mathrm{var}(\mathbf{T}) \cdot e_1(\mathbf{T}{:}1)) + (\overline{\mathrm{var}(\mathbf{T})} \cdot e_1(\mathbf{T}{:}0)))) & \text{otherwise.} \end{cases}$$

The first obligation is to show that $e_1(\mathbf{T})$ is a ternary expression for all Ternary Decision Diagrams, $\mathbf{T}$. Clearly the first two rules yield ternary expressions since 0 and 1 are ternary expressions. Now let $\mathbf{T}$ be a Ternary Decision Diagram of minimal root label in $I\!N$, such that $e_1(\mathbf{T})$ is not a ternary expression. Then for all v in $\{0, \Phi, 1\}$, $e_1(\mathbf{T}{:}v)$ is a ternary expression, since either $\mathrm{ord}(\mathbf{T}{:}v) < \mathrm{ord}(\mathbf{T})$ (minimality of $\mathrm{ord}(\mathbf{T})$), or $\mathrm{ord}(\mathbf{T}{:}v) \in \{0, \Phi, 1\}$ (base case, above). Then $e_1(\mathbf{T})$ is a sum of three products of ternary expressions, and is therefore a ternary expression by the definition of ternary expressions (page 6).

Theorem 8 *If* $\mathbf{T}$ *is a Ternary Decision Diagram and* $|\mathbf{T}|$ *is a monotonic function then* $|\mathbf{T}|(x_1, \ldots, x_n) = 1$ *if and only if* $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = 1$.

Proof: If $\mathbf{T} \in \{0, \Phi, 1\}$ then

$$\begin{aligned} |\mathbf{T}|(x_n, \ldots, x_1) = 1 &\Leftrightarrow \mathbf{T} = \mathrm{tdd}(\mathrm{node}(1,1,1,1)), \\ &\Leftrightarrow e_1(\mathbf{T}) = 1, \\ &\Leftrightarrow |e_1(\mathbf{T})|(x_n, \ldots, x_1) = 1. \end{aligned}$$

The situation where $\mathbf{T}$ is not a simple vertex is more complicated. If this is the case we break the theorem into its two directions of logical implication.

1. Prove $|\mathbf{T}|(x_1,\ldots,x_n) = 1$ implies $|e_1(\mathbf{T})|(x_1,\ldots,x_n) = 1$.

   Now let $\mathbf{S}$ be a Ternary Decision Diagram of minimal root label in $I\!N$such that $|\mathbf{S}|(x_1,\ldots,x_n) = 1$ but $|e_1(\mathbf{S})|(x_1,\ldots,x_n) \neq 1$. Now for all $a \in \{0,\Phi,1\}$ we know that $\mathrm{ord}(\mathbf{S}{:}a) < \mathrm{ord}(\mathbf{S})$ by Property 6 of Decision Diagrams and by Theorem 6 we know $|\mathbf{S}{:}a|$ is monotonic. Therefore the minimality of $\mathrm{ord}(\mathbf{S})$ gives us that $|\mathbf{S}{:}a|(x_n,\ldots,x_1) = 1$ implies $|e_1(\mathbf{S}{:}a)|(x_n,\ldots,x_1) = 1$. Now examine the value of $\mathrm{var}(\mathbf{S})$.

   Case $\mathrm{var}(\mathbf{S}) = 1$

$$\begin{aligned} |\mathbf{S}|(x_n,\ldots,x_1) = 1 &\Rightarrow |\mathbf{S}{:}1|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |e_1(\mathbf{S}{:}1)|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |(\mathrm{var}(\mathbf{S})\cdot e_1(\mathbf{S}{:}1))|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |(\mathrm{var}(\mathbf{S})\cdot e_1(\mathbf{S}{:}1)) + y|(x_1,\ldots,x_n) = 1 + |y| \\ &\Rightarrow |e_1(\mathbf{S})|(x_1,\ldots,x_n) = 1, \end{aligned}$$

   where $y = e_1(\mathbf{T}{:}\Phi) + \overline{\mathrm{var}(\mathbf{T})}\cdot e_1(\mathbf{T}{:}0)$.

   Case $\mathrm{var}(\mathbf{S}) = 0$

$$\begin{aligned} |\mathbf{S}|(x_n,\ldots,x_1) = 1 &\Rightarrow |\mathbf{S}{:}0|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |e_1(\mathbf{S}{:}0)|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |(\overline{\mathrm{var}(\mathbf{S})}\cdot e_1(\mathbf{S}{:}0))|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |(\overline{\mathrm{var}(\mathbf{S})}\cdot e_1(\mathbf{S}{:}0)) + y|(x_1,\ldots,x_n) = 1 + |y| \\ &\Rightarrow |e_1(\mathbf{S})|(x_1,\ldots,x_n) = 1, \end{aligned}$$

   where $y = e_1(\mathbf{T}{:}\Phi) + \mathrm{var}(\mathbf{T})\cdot e_1(\mathbf{T}{:}1)$.

   Case $\mathrm{var}(\mathbf{S}) = \Phi$

$$\begin{aligned} |\mathbf{S}|(x_n,\ldots,x_1) = 1 &\Rightarrow |\mathbf{S}{:}\Phi|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |e_1(\mathbf{S}{:}\Phi)|(x_1,\ldots,x_n) = 1 \\ &\Rightarrow |e_1(\mathbf{S}{:}\Phi) + y|(x_1,\ldots,x_n) = 1 + |y| \\ &\Rightarrow |e_1(\mathbf{S})|(x_1,\ldots,x_n) = 1, \end{aligned}$$

where $y = \overline{\mathrm{var}(\mathbf{T})} \cdot e_1(\mathbf{T}{:}0) + \mathrm{var}(\mathbf{T}) \cdot e_1(\mathbf{T}{:}0)$.

Thus we can not have such an $\mathbf{S}$, and we conclude

$$|\mathbf{T}|(x_n, \ldots, x_1) = 1 \;\Rightarrow\; |e_1(\mathbf{S})|(x_1, \ldots, x_n) = 1.$$

2. Prove $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = 1$ implies $|\mathbf{T}|(x_n, \ldots, x_1) = 1$

   Now let $\mathbf{S}$ be the Ternary Decision Diagram of minimal natural root label such that $|e_1(\mathbf{S})|(x_n, \ldots, x_1) = 1$ but $|\mathbf{S}|(x_1, \ldots, x_n) \neq 1$ and $|\mathbf{S}|$ monotonic. Now we know that

$$\begin{aligned} &|e_1(\mathbf{S}{:}\Phi) + \mathrm{var}(\mathbf{S}) \cdot e_1(\mathbf{S}{:}1) + \overline{\mathrm{var}(\mathbf{S})} \cdot e_1(\mathbf{S}{:}0)| = \\ &\qquad |e_1(\mathbf{S}{:}\Phi)| + |\mathrm{var}(\mathbf{S}) \cdot e_1(\mathbf{S}{:}1)| + |\overline{\mathrm{var}(\mathbf{S})} \cdot e_1(\mathbf{S}{:}0)| \end{aligned}$$

   So if $|e_1(\mathbf{S})|(x_1, \ldots, x_n) = 1$ then one (or more) of the following hold

   1. $$|e_1(\mathbf{S}{:}\Phi)|(x_1, \ldots, x_n) = 1;$$
   2. $$|\mathrm{var}(\mathbf{S}) \cdot e_1(\mathbf{S}{:}1)|(x_1, \ldots, x_n) = 1;$$
   3. $$|\overline{\mathrm{var}(\mathbf{S})} \cdot e_1(\mathbf{S}{:}0)|(x_1, \ldots, x_n) = 1.$$

   Case $|e_1(\mathbf{S}{:}\Phi)|(x_1, \ldots, x_n) = 1$.

   Because $\mathbf{S}$ was minimal we know that

$$|\mathbf{S}{:}\Phi|(x_1, \ldots, x_n) = 1;$$

   but

$$|\mathbf{S}{:}\Phi|(x_1, \ldots, x_n) = |\mathbf{S}{:}\Phi|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})-1}),$$

   and more importantly

$$|\mathbf{S}{:}\Phi|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})} = |\mathbf{S}|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})-1}, \Phi).$$

   Since $|\mathbf{S}|$ is monotonic

$$|\mathbf{S}|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})-1}, \Phi) = 1 \Rightarrow |\mathbf{S}|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})}) = 1.$$

   The extension property (Property 4) gives us

$$|\mathbf{S}|(x_1, \ldots, x_n) = 1.$$

   This contradicts the condition on $\mathbf{S}$'s selection, and so no such $\mathbf{S}$ can exist.

Case $|\mathrm{var}(\mathbf{S}) \cdot e_1(\mathbf{S}{:}1)|(x_1, \ldots, x_n) = 1$.

This implies that
$$|\mathrm{var}(\mathbf{S})|(x_1, \ldots, x_n) = 1$$
and hence $x_{\mathrm{ord}(\mathbf{S})} = 1$. But then by the definition of $|\ |$
$$|\mathbf{S}|(x_1, \ldots, x_n) = |\mathbf{S}{:}1|(x_1, \ldots, x_{\mathrm{ord}(\mathbf{S})-1}).$$

We now recall that
$$|\mathrm{var}(\mathbf{S}) \cdot e_1(\mathbf{S}{:}1)|(x_1, \ldots, x_n) = 1$$
also implies that
$$|e_1(\mathbf{S}{:}1)|(x_1, \ldots, x_n) = 1.$$

But $\mathrm{ord}(\mathbf{S}{:}1) < \mathrm{ord}(\mathbf{S})$; so we know that
$$|\mathbf{S}{:}1|(x_1, \ldots, x_n) = 1.$$

Combining with the previous result we have
$$|\mathbf{S}|(x_1, \ldots, x_n) = 1.$$

Case $|\overline{\mathrm{var}(\mathbf{S})} \cdot e_1(\mathbf{S}{:}0)|(x_1, \ldots, x_n) = 1$.

This implies that
$$|\overline{\mathrm{var}(\mathbf{S})}|(x_1, \ldots, x_n) = 1$$
and hence $x_{\mathrm{ord}(\mathbf{S})} = 0$. Then as in the previous case we obtain
$$|\mathbf{S}{:}0|(x_1, \ldots, x_n) = |\mathbf{S}|(x_1, \ldots, x_n) = 1.$$

Now we have a contradiction in all three cases, since $\mathbf{S}$ was chosen so that
$$|\mathbf{S}|(x_1, \ldots, x_n) \neq 1.$$

Hence no such $\mathbf{S}$ can exist. Thus for all Ternary Decision Diagrams $\mathbf{T}$ which represent monotonic functions
$$|\mathbf{T}|(x_1, \ldots, x_n) = 1 \text{ if and only if } |e_1(\mathbf{T})|(x_1, \ldots, x_n) = 1.$$

$\square$

Theorem 9 *If* $\mathbf{T}$ *is a Ternary Decision Diagram which represents a monotonic function then* $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$ *implies that there is some* $(y_1, \ldots, y_n)$ *such that* $(y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n)$, *and* $|\mathbf{T}|(y_1, \ldots, y_n) = 1$.

Proof:

The theorem is satisfied vacuously for $\mathrm{ord}(\mathbf{T}) \in \{0, \Phi, 1\}$, since $e_1(\{0, \Phi, 1\}) = \{0, 1\}$, $|0|(x_1, \ldots, x_n) \neq \Phi$, and $|1|(x_1, \ldots, x_n) \neq \Phi$ for any ternary vectors $(x_1, \ldots, x_n)$.

Now we make an induction argument on $\mathrm{ord}(\mathbf{T})$, and so must consider a base case. Here we consider $\mathrm{ord}(\mathbf{T}) = 1_{\mathbb{N}}$. Then for all $a \in \{0, \Phi, 1\}$ we know that $\mathrm{ord}(\mathbf{T}{:}a) \in \{0, \Phi, 1\}$, and hence $e_1(\mathrm{ord}(\mathbf{T}{:}a)) \in \{0, 1\}$. Then we know that $e_1(\mathbf{T})(x_1) = \Phi$ implies that $e_1(\mathrm{ord}(\mathbf{T}{:}a))(x_1) = 0$ since otherwise $e_1(\mathbf{T})(x_1) = 1$. Then $e_1(\mathbf{T}) \in \{x_1 + \overline{x_1}, x_1, \overline{x_1}, 0\}$, but clearly since $e_1(\mathbf{T})(x_1) = \Phi$ then $e_1(\mathbf{T}) \neq 0$. We then have:

if $e_1(\mathbf{T}) = x_1$, then $|e_1(\mathbf{T})|(1) = 1$;
if $e_1(\mathbf{T}) = \overline{x_1}$, then $|e_1(\mathbf{T})|(0) = 1$;
if $e_1(\mathbf{T}) = x_1 + \overline{x_1}$, then $|e_1(\mathbf{T})|(0) = 1$.

Therefore there is some y_1 such that $|e_1(\mathbf{T})|(y_1) = 1$, and since $x_1 = \Phi$ then $(y_1) \sqsubseteq (x_1)$. Now assume for all monotonic Ternary Decision Diagrams with $\mathrm{ord}(\mathbf{T}) < N$ for some $N \in \mathbb{N}$ $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$ implies

$$\exists (y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n) \text{ such that } |\mathbf{T}|(y_1, \ldots, y_n) = 1.$$

Now let $\mathbf{T}$ be a Ternary Decision Diagram with $|\mathbf{T}|$ monotonic, and $\mathrm{ord}(\mathbf{T}) \in \mathbb{N}$, $\mathrm{ord}(\mathbf{T}) = N$. Now examine the situation where $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$. Then we know at least one of the following is true

Case $|\mathrm{var}(\mathbf{T}) \cdot e_1(\mathbf{T}{:}1)|(x_1, \ldots, x_n) = \Phi$.

One of the following must then hold.

- $|\mathrm{var}(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$ Then we let

$$(z_1, \ldots, z_n) = (x_1, \ldots, x_{\mathrm{ord}(\mathbf{T})-1}, 1, x_{\mathrm{ord}(\mathbf{T})+1}, \ldots, x_n)$$

  and note $(z_1, \ldots, z_n) \sqsubseteq (x_1, \ldots, x_n)$.
  If $|e_1(\mathbf{T}{:}1)|(z_1, \ldots, z_n) = \Phi$, then the induction hypothesis implies that there is some

$$(y_1, \ldots, y_n) \sqsubseteq (z_1, \ldots, z_n)$$

such that

$$|\mathbf{T}{:}1|(y_1, \ldots, y_n) = 1.$$

But $z_{\mathrm{ord}(\mathbf{T})} = 1$; thus $y_{\mathrm{ord}(\mathbf{T})} = 1$ and so

$$|\mathbf{T}|(y_1, \ldots, y_n) = 1.$$

Otherwise $|e_1(\mathbf{T}{:}1)|(z_1, \ldots, z_n) = 1$. Hence, by Theorem 8, $|\mathbf{T}{:}1|(z_1, \ldots, z_n) = 1$. Since $z_{\mathrm{ord}(\mathbf{T})} = 1$ we have $|\mathbf{T}|(z_1, \ldots, z_n) = 1$.

Thus there is some $(y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n)$ such that

$$|\mathbf{T}|(y_1, \ldots, y_n) = 1.$$

- $|\mathrm{var}(\mathbf{T})|(x_1, \ldots, x_n) = 1$.
  Then $x_{\mathrm{ord}(\mathbf{T})} = 1$ and

  $$|e_1(\mathbf{T}{:}1)|(x_1, \ldots, x_n) = \Phi$$

  The induction hypothesis gives us $(y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n)$, such that $|\mathbf{T}{:}1|(y_1, \ldots, y_n) = 1$. Then, as above, $y_{\mathrm{ord}(\mathbf{T})} = 1$, so $|\mathbf{T}|(y_1, \ldots, y_n) = 1$.

Case $|\overline{\mathrm{var}(\mathbf{T})}{\cdot}e_1(\mathbf{T}{:}0)|(x_1, \ldots, x_n) = \Phi$

We repeat the above argument, replacing the deductions about elements of the vectors being 1 with them being 0 and again find a $(y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n)$ such that

$$|\mathbf{T}|(y_1, \ldots, y_n) = 1.$$

Case $|e_1(\mathbf{T}{:}\Phi)|(x_1, \ldots, x_n) = \Phi$.

We immediately appeal to the induction hypothesis, for $\mathrm{ord}(\mathbf{T}{:}\Phi) < N$, and so we find a $(z_1, \ldots, z_n) \sqsubseteq (x_1, \ldots, x_n)$ such that $|\mathbf{T}{:}\Phi|(z_1, \ldots, z_n) = 1$. Define

$$(y_1, \ldots, y_n) = (z_1, \ldots, z_{\mathrm{ord}(\mathbf{T})-1}, \Phi, z_{\mathrm{ord}(\mathbf{T})+1}, \ldots, z_n).$$

Then $|\mathbf{T}{:}\Phi|(y_1, \ldots, y_n) \sqsubseteq |\mathbf{T}{:}\Phi|(z_1, \ldots, z_n)$ and $|\mathbf{T}|(y_1, \ldots, y_n) \sqsubseteq 1$; so $|\mathbf{T}|(y_1, \ldots, y_n) = 1$.

Thus there is some $(y_1, \ldots, y_n) \sqsubseteq (x_1, \ldots, x_n)$ such that $|\mathbf{T}|(y_1, \ldots, y_n) = 1$. $\square$

Theorem 10 *If* $\mathbf{T}$ *is a Ternary Decision Diagram and* $|\mathbf{T}|$ *is monotonic then* $|e_1(\mathbf{T})|(x_1,\ldots,x_n) = \Phi$ *implies* $|\mathbf{T}|(x_1,\ldots,x_n) = \Phi$.

Proof: By Theorem 9, if $|e_1(\mathbf{T})|(x_1,\ldots,x_n) = \Phi$, then there is some $(y_1,\ldots,y_n) \sqsubseteq (x_1,\ldots,x_n)$, such that $|\mathbf{T}|(y_1,\ldots,y_n) = 1$. Since $|\mathbf{T}|$ is monotonic, $|\mathbf{T}|(y_1,\ldots,y_n) \sqsubseteq |\mathbf{T}|(x_1,\ldots,x_n)$; thus $|\mathbf{T}|(x_1,\ldots,x_n) \in \{\Phi, 1\}$.

We know $|\mathbf{T}|(x_1,\ldots,x_n) \neq 1$ by Theorem 8; since $|e_1(\mathbf{T})|(x_1,\ldots,x_n) \neq 1$. We therefore conclude that $|\mathbf{T}|(x_1,\ldots,x_n) = \Phi$, as required. □

This could also be stated as the contrapositive, namely that if $|\mathbf{T}|(x_1,\ldots,x_n) \neq \Phi$ then $|e_1(\mathbf{T})|(x_1,\ldots,x_n) \neq \Phi$. It is unfortunately not the case, however, that the converse is true. There are Ternary Decision Diagrams for which $|\mathbf{T}|$ is monotonic and $|\mathbf{T}|(x_1,\ldots,x_n) = \Phi$ but $|e_1(\mathbf{T})|(x_1,\ldots,x_n) = 0$. The simplest such case is $\mathrm{tdd}(\Phi)$. We now define a mapping from Ternary Decision Diagrams to expressions that will pick up any Φ values that were missed. Formally we write

$$e_\Phi(\mathbf{T}) = \begin{cases} 0, & \text{if } \mathbf{T} \in \{0,1\}, \\ \Phi, & \text{if } \mathbf{T} = \Phi, \\ (\overline{\mathrm{var}(\mathbf{T})}\cdot\mathrm{var}(\mathbf{T})\cdot e_\Phi(\mathbf{T}{:}\Phi) + ((\mathrm{var}(\mathbf{T})\cdot e_\Phi(\mathbf{T}{:}1)) + (\overline{\mathrm{var}(\mathbf{T})}\cdot e_\Phi(\mathbf{T}{:}0)))), & \\ & \text{otherwise.} \end{cases}$$

We then make the observation that the function represented by this expression is never 1.

Theorem 11 *Let* $\mathbf{T}$ *be a Ternary Decision Diagram which represents a monotonic function; then* $|e_\Phi(\mathbf{T})|(x_1,\ldots,x_n) = \Phi$ *if and only if* $|\mathbf{T}|(x_1,\ldots,x_n) = \Phi$.

Proof: If $\mathrm{ord}(\mathbf{S})$ is in $\{0, \Phi, 1\}$ then

$$\begin{aligned} |e_\Phi(\mathbf{T})|(x_1,\ldots,x_n) = \Phi &\Leftrightarrow \mathrm{ord}(\mathbf{T}) = \Phi \\ &\Leftrightarrow |\mathbf{T}|(x_1,\ldots,x_n) = \Phi \end{aligned}$$

Let $\mathbf{S}$ be a Ternary Decision Diagram which represents a monotonic function, and for which $\mathrm{ord}(\mathbf{S})$ is minimal amongst all Ternary Decision Diagrams such that $\mathrm{ord}(\mathbf{S}) \in I\!N$ and $|\mathbf{S}|(x_1, \ldots, x_n) = \Phi$ but $|e_\Phi(\mathbf{S})|(x_1, \ldots, x_n) \neq \Phi$. By the previous observation $|e_\Phi(\mathbf{S})|(x_1, \ldots, x_n) = 0$. Now consider the possible values for $\mathrm{var}(\mathbf{S})$.

$\mathrm{var}(\mathbf{S}) = 0$

Then

$$\begin{aligned} |e_\Phi(\mathbf{S})|(x_1, \ldots, x_n) &= |0 + e_\Phi(\mathbf{S}{:}0) + 0|(x_1, \ldots, x_n) \\ &= |e_\Phi(\mathbf{S}{:}0)|(x_1, \ldots, x_n) \\ &= 0 \end{aligned}$$

But $|\mathbf{S}{:}0|(x_1, \ldots, x_n) = |\mathbf{S}|(x_1, \ldots, x_n) = \Phi$; thus $|e_\Phi(\mathbf{S}{:}0)|(x_1, \ldots, x_n) = \Phi$ and hence $\mathbf{S}$ was not minimal.

$\mathrm{var}(\mathbf{S}) = 1$

As in the previous argument, this implies $|e_\Phi(\mathbf{S}{:}1)|(x_1, \ldots, x_n) = 0$, which contradicts the minimality of $\mathbf{S}$.

$\mathrm{var}(\mathbf{S}) = \Phi$

Substituting the value Φ for $\mathrm{var}(\mathbf{S})$ in the formula for $e_\Phi(\mathbf{S})$ gives us that $|e_\Phi(\mathbf{S}{:}v)|(x_1, \ldots, x_n) = 0$ for all $v \in \{0, \Phi, 1\}$. In particular $\mathbf{S}{:}\Phi$ violates the minimality of $\mathbf{S}$, and so this can not be the case either.

Thus we have exhausted all possibilities and are forced to conclude that there are in fact no Ternary Decision Diagrams such that $|\mathbf{T}|(x_1, \ldots, x_n) = \Phi$ but $|e_\Phi(\mathbf{T})|(x_1, \ldots, x_n) \neq \Phi$. We must now show that $|e_\Phi(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$ implies $|\mathbf{T}|(x_1, \ldots, x_n)$.

Let $\mathbf{S}$ be a Ternary Decision Diagram of minimal root label such that $|e_\Phi(\mathbf{S})|(x_1, \ldots, x_n) = \Phi$ but $|\mathbf{S}|(x_1, \ldots, x_n) \neq \Phi$. Again we examine $\mathrm{var}(\mathbf{S})$:

$\mathrm{var}(\mathbf{S}) = 0$

Then $e_\Phi(\mathbf{S})$ can be simplified to $e_\Phi(\mathbf{S}{:}0)$, and thus by the minimality of $\mathbf{S}$ we have $|\mathbf{S}|(x_1, \ldots, x_n) = \Phi$.

$\mathrm{var}(\mathbf{S}) = 1$

As above $e_\Phi(\mathbf{S})$ can be simplified to $e_\Phi(\mathbf{S}{:}1)$, and thus by the minimality of $\mathbf{S}$ we have $|\mathbf{S}|(x_1, \ldots, x_n) = \Phi$.

$\text{var}(\mathbf{S}) = \Phi$

If $\text{var}(\mathbf{S}) = \Phi$ and $|e_{\Phi}(\mathbf{S})|(x_1, \ldots, x_n) = \Phi$ then at least one of $|e_{\Phi}(\mathbf{S}{:}a)|(x_1, \ldots, x_n) = \Phi$. Therefore, by minimality of $\mathbf{S}$, there is some a such that $|\mathbf{S}{:}a|(x_1, \ldots, x_n) = \Phi$. If $a \in \{0, 1\}$ then $|\mathbf{S}{:}\Phi|(x_1, \ldots, x_n) = \Phi$ by monotonicity, otherwise $|\mathbf{S}{:}\Phi|(x_1, \ldots, x_n) = \Phi$ because $a = \Phi$. But $\text{var}(\mathbf{S}) = \Phi$; hence $|\mathbf{S}|(x_1, \ldots, x_n) = \Phi$ by definition of $|\ \ |$.

Now we have shown that, for monotonic Ternary Decision Diagrams, $\mathbf{T}$,

$$|e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = \Phi \text{ if and only if } |\mathbf{T}|(x_1, \ldots, x_n) = \Phi.$$

□

Theorem 12 *Let* $\mathbf{T}$ *be a Ternary Decision Diagram which represents a monotonic function; then* $|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})| = |\mathbf{T}|$.

Proof: Let $(x_1, \ldots, x_n)$ be in $\{0, \Phi, 1\}^n$; consider $|\mathbf{T}|(x_1, \ldots, x_n)$.

$|\mathbf{T}|(x_1, \ldots, x_n) = 1$

Then $|e_1(\mathbf{T})|(x_1, \ldots, x_n) = 1$, by Theorem 8; therefore

$$|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = 1.$$

$|\mathbf{T}|(x_1, \ldots, x_n) = \Phi$

Then $|e_1(\mathbf{T})|(x_1, \ldots, x_n) \neq 1$ by Theorem 8, and by Theorem 11, $|e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$. So $|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = \Phi$.

$|\mathbf{T}|(x_1, \ldots, x_n) = 0$

Then

$$\begin{array}{rll} & |e_1(\mathbf{T})|(x_1, \ldots, x_n) \neq 1 & \text{by Theorem 8,} \\ & |e_1(\mathbf{T})|(x_1, \ldots, x_n) \neq \Phi & \text{by Theorem 10,} \\ & |e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) \neq \Phi & \text{by Theorem 11,} \\ \text{and} & |e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) \neq 1 & \text{by observation,} \end{array}$$

so $|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = 0$.

Thus $|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})|(x_1, \ldots, x_n) = |\mathbf{T}|(x_1, \ldots, x_n)$ and so

$$|e_1(\mathbf{T}) + e_{\Phi}(\mathbf{T})| = |\mathbf{T}|.$$

□

Chapter 4

Implementation

4.1 Implementation Platform

The T.E.R.M. program is implemented in the **C** programming language, using the UNIX operating system. It has been successfully compiled under Ultrix 3.1 (VAX), Ultrix 4.2 (MIPS), Irix 4.0 (SGI) and SunOS 4.1 (SPARC) operating systems. The main user interface and parser is built using `yacc` and `lex`. It is approximately 350 lines of code. The approximate sizes of the individual modules is given with their descriptions.

Module	lines	bytes of object code
Literal Sets	628	9688
Expression Trees	858	21540
Result Caches	120	2672
Ternary Decision Diagrams	566	12016
Sets of Literal Sets (terms)	437	7000
Mainline	61	2520

Table 4.1: Code size for various components of T.E.R.M.

4.2 Implementation Parts

The T.E.R.M. program is implemented as a set of modules which are responsible for providing operators on certain abstract data types, or objects.

4.2.1 Result Caches

A result cache is a mechanism for reducing recomputation. There are three basic operations on a result cache: assertion, query, and removal. The assertion operator specifies the desired result of a future query. After an assertion of a (domain, range) pair a query on that domain value will result in the return of the range value, unless a removal of one of the values of the domain or range has occurred since the assertion, but before the query.

A relaxation of this requirement is that in a bounded result cache the query of a function may return `NONE` instead of a valid y if space requirements of its storage are exceeded. The cache is only useful if its operations are significantly quicker than the computation of the function it is caching. The current implementation is not great, but seems to help. The T.E.R.M. program uses two result caches with x_i, y being integers and $n = 2$, $m = 1$.

A similar mechanism exists in the more general computer algebra package MAPLE [11], called "remember tables". In MAPLE the remember table is explicitly connected with a single function or procedure, and is used by the invocation mechanism. Here we are using a simpler invocation mechanism, and the function performs its own lookup after checking for trivial results.

4.2.2 Literal Sets

A literal set is used to represent a subset of $V \times \{\text{normal}, \text{inverted}\}$, where V represents a fixed set of variables. The operations allowed are

empty Creates an empty set.

insert, remove Change the contents of a set.

element of Determines if a given pair (v, s) is in the referenced set.

subset of, equal to Determine inclusion relationships between sets.

copy, to string, to expression Forms a new item representing the set, either a (distinct) set with the same contents, a text representation, or an expression tree of the product of the elements.

size Returns the number of elements of the set.

This list most notably misses the complement, union, and intersection operators. These were not implemented as they were not required. They probably should be included for more general applications.

4.2.3 Sets of Literal Sets (terms)

The next class of object is a set of the previously described sets of literals. These are logically described as sets of terms. The following operations are supported

empty, copy Creates a new set, with the initial contents either empty or the same as the copied set.

add, union These calls modify the set by either inserting a new element or a whole set of new elements.

covered Determines if some element of the set is a subset of the desired element. This rather strange membership test turns out to be the one we want when we use these objects to represent sums of products.

simplify This operation reduces the given set in such a way as to minimize the number of elements while maintaining the same coverage as defined above. This is used for elimination of redundant terms.

to text This produces a string of text where the elements are separated by the desired strings, and themselves converted by the underlying set of literals to text function.

As above, a specialized set of operators has been implemented. More general and usual operators could not be easily added if necessary.

4.2.4 tdds

tdds are the implemented form of Ternary Decision Diagrams. The intention of the code is that tdds have the same properties as Ternary Decision Diagrams. The following operators are provided.

zero, one, phi Three constants are provided to represent the terminal vertices $\{0,\ \Phi,\ 1\}$, or, more precisely, node(0,0,0,0), node(1,1,1,1), and node(Φ,Φ,Φ,Φ).

make The general purpose operator which forms the tdd for given node(l,x,y,z) is provided, where x, y, z must be pre-existing tdds. This operation will take care that such objects are unique, and irredundant at the first level, i.e. node(l,x,x,x) $= x$, and there is only ever one vertex for any particular (l, x, y, z).

and,or,not The three operations on tdds are implemented as direct calls to the library. These use the knowledge that the domain of the tdds obeys the laws of ternary algebra.

print This operation will cause a text representation of the tdd to be output. This operation would be more useful if the output was available to be used in further computations, but was not implemented this way.

4.2.5 Expression Trees

These objects are used to represent expressions, and also axioms. They represent the form of the expression as well as its value. They are maintained as well as the functional representation to allow the user to follow, verify and use the computed results in a natural way. The following operators are provided:

zero, one, phi, literal These basic constructors will return new objects which represent the expressions $0, 1, \Phi, x_1, \ldots, x_n$.

not, and, or These will return objects representing $\overline{F}$, $F{+}E$, and $F{\cdot}E$ respectively.

equivalence This operator determines if the two given expressions are equivalent. This operation is a constant time operation as the tdd for each expression is already available.

do trivial simplifications This rather nebulous operator will do all simplifications that are blatantly and painfully obvious to every user of the system. Currently I feel that no operators fall into this class as a large number of changes may produce a result much simpler than a naïve user might believe. Future implementations may include recursive applications of DeMorgan's laws to move all negation to the lowest terms.

print as tree, print as expression, print tdd These routines cause the various different ways of representing the expression to be printed. As with tdds, these operators should be replaced by ones which convert to text representation. This has not been done because of issues involved in the allocation of storage for such items in a robust system independent manner. This issue should be resolved, but again it would produce no discernible difference in the result.

replace a subtree This operations replaces a given subtree of an expression with another subtree. This may or may not change the value represented by the expression. The user is responsible for ensuring that the subtrees are equivalent. If they are not, the algorithms may stop producing correct results. This should be handled in the next version.

define an axiom/rule This allows a pattern or axiom to be defined for later use. The axiom is specified in the form of two expression trees, where their interpretation is altered so that a variable in a rule tree (literal) may be bound to any single expression tree, but all such bindings are unique. The axiom is also uni-directional, and its interpretation is that "something like tree A" may be replaced by "something like tree B" where the same bindings apply to the replacement. See 4.4 for more details.

apply an axiom/rule This operation performs a pattern match and replacement using the given form of equivalence.

bind a literal, clear bindings This allows control over the binding of literals to actual expressions in the application of a rule. For more description of this process see 4.4. A single literal may be given an expression tree, or all current bindings may be released. These are required to introduce new parts to subexpressions easily, e.g. applying $(a) \rightarrow ((a) + 0)$

print rules The set of all defined rules, their names and values may be printed. Again this should be replaced by converting to a text representation, but the representation would be prohibitively long for non-trivial examples.

4.3 Interface

The interface of the T.E.R.M. program is described in a separate user's manual, [15].

4.4 Axiom Application

Axioms are statements of equality. Axioms are usually stated as universally quantified over the entire algebra system. However, the values quantified must have a single binding throughout.

We will represent axioms as explicit equivalences between expressions. These equivalences will be used in a unidirectional, transform-like way. By this we mean that an axiom of the form $a{+}a = a$ will be stored in two forms, $a{+}a \mapsto a$, and $a \mapsto a{+}a$. The application of an axiom will proceed in three steps. The first is the binding/unification step. This is where an appropriate value for all literals present on the left side of and axiom (e.g. a) have values chosen from the space of all expression trees. The second step is to build an equivalent expression tree from the right side of the axiom, using the same binding as before. The third step is to replace the subtree matched by the subtree built. These steps can be done as follows

```
tree_bind( expression tree t, expression tree a )
case a in
        0, 1, Φ       return equiv(t,a)
        x1 ... xn     if (bound(a)) then
                              return(equiv(t,val(a)))
                      else
                              literal_bind(a,t)
                              return true
                      endif
        ¯(a')         if (t = ¯(t')) then
                              return(tree_bind(t',a'))
                      else
                              return false
                      endif
        +(a', a'')    if (t = +(t', t'')) then
                              return(tree_bind(t',a')∧tree_bind(t'',a''))
                      else
                              return false
                      endif
        ·(a', a'')    if (t = ·(t', t'')) then
                              return(tree_bind(t',a')∧tree_bind(t'',a''))
                      else
                              return false
                      endif
endcase
```

Algorithm 1: Binding a rule to an expression

Chapter 5

Conclusions

The implementation of this program has provided a useful tool, as well as exposed areas of further study.

5.1 Efficiency

The method of representing ternary functions by a graph structure has proved to be feasible with commonly available machines. The efficiency is adequate for use as an interactive tool for moderate size examples. In practice we have found the entry and reading of commands to be the dominant time in our uses of the tool. As an informal example: in use for some testing we generate almost all of the expressions of 2 variables, using 6 or fewer operators, and ask T.E.R.M. to produce the expressions from the resulting Ternary Decision Diagrams. We find that it processed the 134031 expressions in 114 CPU seconds on a 25MHz R3000 based machine, or 248 CPU seconds on a SPARC IPC.

5.2 Future Work

While many of the questions, regarding Ternary Algebras, have already been answered there are several areas that are open to extensions, and more in-depth investigation.

Conversion Algorithm

The process of converting a Ternary Decision Diagram to an expression representation involves twin traversals of the diagram. Worse still, the traversal as it stands takes no advantage of having already visited a vertex in the diagram. Investigating methods for creating an expression is one area that we see as a source of interesting potential for rewarding advances.

Minimization

The properties of the canonic expression derived from the procedure are not fully investigated or discussed. For example, we believe that for a Ternary extension the expression returned will be a minimal sum of products form of the function after simple application of the removal of redundant terms. We note that more complex terms than necessary arise when a ternary expression which involving Φ is returned. These could be studied for both completeness' and interest's sake.

Reduction

The axiomatic reduction of expressions is currently completely under the user's direction. There is a probability that many of the simple reductions and generally useful transformations can be automated, especially if the original expression comes from another program, such as a circuit analysis package, or a specification compiler.

Interface

The interface to the tool is designed to be usable, but it is far from intuitive. A better way for a user to direct manipulations would be interesting and beneficial. The interface to another program could be improved, and the issues of function specification studied.

Dual Conversion

The entire conversion from a Ternary Decision Diagram uses a product dominant, searching for 1s approach. A dual approach should exist in which a traversal searches for 0s and a product of sums form is found. We see no great advantage to doing this, but it is equally valid. Our development of the sum of products form was motivated by personal preference and bias, not because of a perceived benefit.

Variable Ordering

Some authors have considered the effects of the selection of the order of the variable on the resulting Decision Diagrams in the Boolean case. Most of the work has a direct, simple analogue in Ternary Decision Diagrams. This work was also intended as a tool for understanding some of the properties of Ternary Decision Diagrams, and so a predictable order was chosen. Efficient methods for selecting, specifying and manipulating the order could prove an interesting topic and an enhancement of the abilities of the program.

Attributed Edges

Some authors[16] have investigated the use of different kinds of edges to represent modifications to the interpretation of the children of a vertex, based on how the child was reached. The simplest examples of such edges are complement edges and variable shift edges. These edges are more complex to store, but can improve the subgraph sharing of Ordered Binary Decision Diagrams significantly. The ternary analogues and extensions of this idea have not been investigated. We believe the unique representation is going to be more difficult to guarantee in a Ternary system. We also note that the *ternary-valued* Binary Decision Diagrams considered by Minato et al. are different from our Ternary Decision Diagrams because we consider our domain to be ternary, where Minato et al. consider only the range to be ternary. Under that assumption the important difference between $a + \overline{a}$ and 1 is lost, since $a + \overline{a} = 1$ for all binary values.

Bibliography

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient Implementation of a BDD Package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[2] R. E. Bryant. A Switch-level Model and Simulator for MOS Digital Systems. *IEEE Transactions on Computers*, C-33(2):160–177, 1984.

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

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

[5] J. A. Brzozowski and C-J. H. Seger. Advances in Asynchronous Circuit Theory - Part I: Gate and Unbounded Inertial Delay Models. *Bulletin of the European Association for Theoretical Computer Science*, 1(42):198–249, 1990.

[6] J. A. Brzozowski and C-J. H. Seger. Advances in Asynchronous Circuit Theory - Part II: Bounded Inertial Delay Models, MOS Circuits, Design Techniques. *Bulletin of the European Association for Theoretical Computer Science*, 1(43):199–263, 1991.

[7] J. A. Brzozowski and C-J. H. Seger. *Asynchronous Networks.* Forthcoming, 1994.

[8] J. A. Brzozowski and M. Yoeli. On a Ternary Model of Gate Networks. *IEEE Transactions on Computers*, C-28(3):178–184, 1979.

[9] J. A. Brzozowski and M. Yoeli. Combinational and static CMOS Networks. *Integration, The VLSI Journal*, 5:103–122, 1987.

[10] J. R. Burch, E. M. Clarke, McMillan K. L., and D. L. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *27th ACM/IEEE Design Automation Conference*, pages 46–51, 1990.

[11] B. W. Char, K. O. Geddes, G. H. Gonnet, M. B. Monagan, and S. M. Watt. *MAPLE Reference Manual.* WATCOM Publications, 5 edition, 1988.

[12] M. Goto. Application of Logical Mathematics to the Theory of Relay Networks. *Journal of the IEE Japan*, 69, 1949.

[13] K. Karplus. Using If-Then-Else DAGs for Multi-Level Logic Minimization. In C. Seitz, editor, *Advance Research in VLSI*, pages 101–118. MIT Press, 1989.

[14] J-C. Madre and J-P. Billon. Proving Circuit Correctnes using Formal Comparison Between Expected and Extracted Behaviour. In *25th ACM/IEEE Design Automation Conference*, pages 52–57, 1988.

[15] P. B. G. Mayo. T.E.R.M. User's Reference Guide, 1993.

[16] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagrams with Attributed Edges for Efficient Boolean Function Manipulation. In *27th ACM/IEEE Design Automation Conference*, pages 52–57, 1990.

[17] M. Mukaidono. Regular Ternary Logic Functions - Ternary Logic Functions Suitable for Treating Ambiguity. In *IEEE Symposium on Multiple-Valued Logic*, pages 286–291, 1983.

[18] M. Mukaidono. Fundamental Properties of Logic Simulation Based on Three-Valued Logic. In *IEEE Workshop on Languages for Automation*, pages 242–245, 1987.

[19] M. Mukaidono and I. G. Rosenberg. k-Valued Functions for Treating Ambiguities: Their Clone and a Normal Form. In *IEEE Symposium on Multiple-Valued Logic*, pages 286–291, 1986.

[20] M. Mukaidono and I. G. Rosenberg. Clones of Isotone Operators for Half-Bounded Orders and Regular Functions. In *IEEE Symposium on Multiple-Valued Logic*, pages 349–355, 1987.

[21] D. S. Reeves and M. J. Irwin. Fast Methods for Switch-Level Verification of MOS Circuits. In *IEEE Transactions on Computer Aided Design*, pages 766–779, 1987.

[22] A. Srinivasan, T. Kam, S. Malik, and R. K. Brayton. Algorithms for Discrete Function Manipulation. In *IEEE International Conference on Computer Aided Design*, pages 92–95, 1990.

[23] M. Yoeli and J. A. Brzozowski. Ternary Simulation of Binary Gate Networks. In Michael J. Dunn and George Epstein, editors, *Modern Uses of Multiple Valued Logic*, pages 41–53. D. Reidel Publishing Company, 1975.

[24] M. Yoeli and S. Rinon. Application of Ternary Algebra to the Study of Static Hazards. *Journal of the ACM*, 11(1):84–97, 1964.