Binary Decision Diagrams by Shared Rewriting

Jaco van de Pol[1] Hans Zantema[2,1]
Jaco.van.de.Pol@cwi.nl hansz@cs.uu.nl

1: CWI, P.O.-box 94.079, 1090 GB, Amsterdam, The Netherlands

2: Department of Computer Science, Utrecht University,
P.O.-box 80.089, 3508 TB Utrecht, The Netherlands

Abstract

BDDs provide an established technique for propositional formula manipulation. In this paper we re-develope the basic BDD theory using standard rewriting techniques. Since a BDD is a DAG instead of a tree we need a notion of shared rewriting and develope appropriate theory. A rewriting system is presented by which canonical ROBDDs can be obtained. For this rewriting system a *layerwise* strategy is proposed having the same time complexity as the traditional algorithm, and a *lazy* strategy sometimes performing much better than the traditional algorithm.

1 Introduction

Equivalence checking and satisfiability testing of propositional formulas are basic but hard problems in many applications, including hardware verification [6] and symbolic model checking [7]. Binary decision diagrams (BDDs) [4, 5, 9], are an established technique for this kind of boolean formula manipulation. The basic ingredient is representing a boolean formula by a unique canonical form, the so called reduced ordered BDD (ROBDD). After canonical forms have been established equivalence checking and satisfiability testing is trivial. Constructing the canonical form however, can be very costly; it is even possible that the size of the canonical form is exponential in the size of the original formula. A main goal of the BDD approach is to keep constructing these canonical forms tractable for as many boolean formulas as possible.

BDDs are recursively defined structures and they are manipulated by repeating small steps. It seems rather natural to view the BDD theory and the manipulations on BDDs from a term rewriting point of view. In this paper, we pursue this view on the following lines: First, a signature for BDDs is given. Next we consider a finite axiomatization of logical equivalence on these trees. Using the fairly standard rewriting techniques ([2]) of critical pair analysis and recursive path ordering, we turn this into a complete, i.e. normalizing and confluent, term rewriting system (TRS), for which the normal forms are exactly the ROBDDs. In this way great part of BDD theory is obtained for free: the existence of an ROBDD representation follows from the normalization property,

and unicity of the ROBDD representation follows from the confluence property. The main theorem that propositional formulas are logically equivalent if and only their ROBDD representations are syntactically equal, turns out to be a corollary of soundness and completeness of the basic axiomatization.

A complication is that the relative efficiency of BDDs hinges on the maximally shared representation. In order to avoid the intricacies of maximally shared graph rewriting, we present an elegant abstraction. Instead of introducing a rewrite relation on graphs, we introduce a *shared rewrite step* on terms. In a shared rewrite step, all identical redexes have to be rewritten at once. We prove that if a TRS is complete, then the shared version is so too. This enables us to develop the main theory in standard term rewriting (without sharing). The rewrite analysis can be lifted to shared rewriting for free. This lifting is needed to study the algorithmic complexity in terms of rewrite steps.

The power of a rewriting approach to BDD theory goes beyond a re-development of existing theory. In particular, we describe a TRS to be used to compute the ROBDD for a propositional formula. Instead of correctness of one single algorithm this implies that every reduction strategy represents a correct algorithm. In this respect we hope that the BDD-world can benefit from the huge amount of research on rewriting strategies. Moreover, in the BDD-world various extensions are emerging, both with respect to the data structure as well as the algorithmics. (see e.g. [1]). Term rewriting can present a general framework for desribing the variations.

After having established the basic theory, we present a TRS for applying logical operations to ROBDDs and prove its correctness. This generalizes the traditional algorithm, using Bryant's function *apply*. Then a *layerwise* reduction strategy for this TRS is given which mimics the usual *apply*-algorithm, and we prove that it has the same time complexity. Finally, a *lazy* strategy is given, with an example in which this performs much better than the traditional algorithm.

In Section 2 we present basic theory for decision trees and describe how canonical forms are obtained by rewriting. In Section 3 we present our approach to shared rewriting, independent of the particular application to BDDs. In Section 4 ROBDDs are presented as shared representations of canonical forms and a TRS is given and analyzed for various strategies to compute them. Finally in Section 5 some conclusions are given.

2 Decision trees

We consider a set A of binary *atoms*, whose typical elements are denoted by $p, q, r, \ldots$. An *instance* s over A is defined to be a map from A to $\{\mathsf{true}, \mathsf{false}\}$; intuitively for an atom p and an instance s the value $s(p)$ represents whether the boolean atom p holds for the instance s or not.

A *binary decision tree* over A is a binary tree in which every internal node is labeled by an atom and every leaf is labeled either true or false. In other words, a decision tree over A is defined to be a ground term over the signature having true and false as constants and elements of A as binary symbols.

Introducing the convention that in a decision tree the left branch of a node

p corresponds to p taking the value true and the right branch corresponds to false, a boolean value $\phi(T,s)$ can be assigned to every decision tree T and every instance s, inductively defined as follows

$$\begin{array}{rcll} \phi(\mathsf{true},s) & = & \mathsf{true} & \\ \phi(\mathsf{false},s) & = & \mathsf{false} & \\ \phi(p(T,U),s) & = & \phi(T,s) & \text{if } s(p)=\mathsf{true} \\ \phi(p(T,U),s) & = & \phi(U,s) & \text{if } s(p)=\mathsf{false}. \end{array}$$

Alternatively, in propositional notation the last two lines can be written as

$$\phi(p(T,U),s) \;=\; (s(p)\wedge\phi(T,s))\vee(\neg s(p)\wedge\phi(U,s)),$$

or equivalently as $\phi(p(T,U),s) \;=\; (s(p)\to\phi(T,s))\wedge(\neg s(p)\to\phi(U,s))$.

The function $\phi(T,-)$ is the boolean function described by T. Conversely, it is not difficult to see that every boolean function on A can be described by a decision tree. One way to do so is building a decision tree such that in every path from the root to a leaf every $p\in A$ occurs exactly once, and plugging the values true and false in the $2^{\#A}$ leaves according to the $2^{\#A}$ lines of the truth table of the given boolean function.

For any decision tree T let $\#(T)$ be the size of T, being the number of internal nodes, defined inductively by

$$\#(\mathsf{true})=\#(\mathsf{false})=0, \#(p(T,U))=1+\#(T)+\#(U).$$

Two decision trees T and U are called *equivalent*, denoted as $T\simeq U$, if they represent the same boolean function, i.e., if

$$\phi(T,s)=\phi(U,s) \;\text{ for all } s:A\to\{\mathsf{true},\mathsf{false}\}.$$

Decision equivalence can be described by an equational axiomatization as follows. Let $\mathcal{E}$ consist of the equations

$$\begin{array}{lrcl} (1) & p(x,x) & = & x \\ (2) & p(q(x,y),q(z,w)) & = & q(p(x,z),p(y,w)) \\ (3) & p(p(x,y),z) & = & p(x,z) \\ (4) & p(x,p(y,z)) & = & p(x,z) \end{array}$$

for all $p,q\in A$. Note that $\mathcal{E}$ is finite if and only if A is finite. Here x,y,z,w are variables from a set X of variable symbols for which decision trees have to be substituted. More precisely, if $\sigma:X\to D$, and t is a term built from atoms from A and variables from X, then t^σ is defined inductively as follows

$$\begin{array}{rcll} x^\sigma & = & \sigma(x) & \text{for all } x\in X, \\ p(t,u)^\sigma & = & p(t^\sigma,u^\sigma) & \text{for all } p\in A. \end{array}$$

Let $\equiv_\mathcal{E}$ be the congruence generated by $\mathcal{E}$, i.e., $\equiv_\mathcal{E}$ is the smallest binary relation on D satisfying

- $t^\sigma\equiv_\mathcal{E} u^\sigma$ for every equation $t=u$ in $\mathcal{E}$ and every $\sigma:X\to D$, and

- $\equiv_{\mathcal{E}}$ is reflexive, symmetric and transitive, and
- if $T \equiv_{\mathcal{E}} U$ then $p(T,V) \equiv_{\mathcal{E}} p(U,V)$ and $p(V,T) \equiv_{\mathcal{E}} p(V,U)$ for all $p \in A$ and all $V \in D$.

We prove that $\mathcal{E}$ is a sound and complete axiomatization for decision equivalence, i.e., the relations $\equiv_{\mathcal{E}}$ and $\simeq$ on decision trees coincide. This means that two decision trees are equivalent if and only if this can be derived by only applying the four types of equations in $\mathcal{E}$. Soundness of $\mathcal{E}$ (the 'if'-part of the statement) is evident; the hard point is completeness. A straightforward elementary proof is given in [11]; here we give a rewrite approach. This rewrite approach does not lead to completeness only, it is also the basis of uniqueness of the ROBDD representation.

The first step is to complete $\mathcal{E}$: find a confluent and terminating rewrite system DT such that $\equiv_{\mathcal{E}}$ and $\leftrightarrow^*_{DT}$ coincide. One problem in doing so is orienting rule (2). If between two atoms p and q no preference is given, this cannot be oriented without getting cyclic reductions. The way to solve this is choosing a total order $<$ on A, and orient the rewrite rules in such a way that the left hand side is greater than the right hand side with respect to the corresponding recursive path order. In this way all equations are oriented from left to right, where equation (2) is only allowed for $q < p$. This rewrite system has non-converging critical pairs, in particular $\langle p(q(x,y),z), q(p(x,z),p(y,z))\rangle$, obtained from rewriting $p(q(x,y),q(z,z))$ by rule (1) and rule (2), respectively. Orienting yields the new set of rewrite rules

$$p(q(x,y),z) \to q(p(x,z),p(y,z))$$

for all p,q satisfying $p > q$, and by symmetry also

$$p(x,q(y,z)) \to q(p(x,y),p(x,z))$$

for all p,q satisfying $p > q$. Surprisingly, the original rule (2) can be removed now since

$$p(q(x,y),q(z,w)) \to^+ q(p(x,z),p(y,w))$$

by only using rules (3), (4) and these new rules. We define the rewrite system DT to consist of the rules

$$\begin{array}{rcll} p(x,x) & \to & x & \text{for all } p \\ p(p(x,y),z) & \to & p(x,z) & \text{for all } p \\ p(x,p(y,z)) & \to & p(x,z) & \text{for all } p \\ p(q(x,y),z) & \to & q(p(x,z),p(y,z)) & \text{for } p > q \\ p(x,q(y,z)) & \to & q(p(x,y),p(x,z)) & \text{for } p > q. \end{array}$$

We have defined DT in such a way that indeed $\equiv_{\mathcal{E}}$ and $\leftrightarrow^*_{DT}$ coincide. Moreover, DT is terminating since every left hand side is greater than the corresponding right hand side with respect to recursive path order. Finally, it can be checked that all critical pairs are convergent. This can be done by hand or automatically, for the latter approach it has to be remarked that it suffices to prove it for the case of $\#A = 3$ since no rule contains more than two different symbols. Since all critical pairs converge DT is locally confluent, and since DT is terminating too we conclude that DT is confluent.

Definition 1 *A decision tree is in canonical form with respect to the order $<$ on A if on every path from the root to a leaf the atoms occur in strictly increasing order, and no subterm of the shape $p(T_1, T_2)$ exists for which T_1 and T_2 are syntactically equal.*

Clearly a decision tree is in canonical form if and only if it is in normal form with respect to DT. Since DT is terminating and confluent we have the following theorem.

Theorem 2 *Every decision tree reduces by DT to a unique canonical form, and T_1 and T_2 have the same canonical form if and only if $T_1 \equiv_{\mathcal{E}} T_2$.*

Next we prove completeness of the equational axiomatization. First we need a lemma.

Lemma 3 *Let T, U be decision trees in canonical form satisfying $T \simeq U$. Then $T = U$.*

Proof: We apply induction on $\#(T) + \#(U)$. If $\#(T) + \#(U) = 0$ then both T and U are true or both T and U are false and we are done.

Consider the case $\#(T) + \#(U) > 0$. In case either T or U is equal to true or false, say T is equal to true, then U can be written as $U = p(U_1, U_2)$. Since U is in canonical form both U_1 and U_2 are in canonical form and p does neither occur in U_1 nor in U_2. Since $U \simeq \mathsf{true}$ we obtain $U_1 \simeq \mathsf{true}$ and $U_2 \simeq \mathsf{true}$; from the induction hypothesis we conclude $U_1 = \mathsf{true} = U_2$, contradicting the assumption that U is in canonical form.

In the remaining case we have $T = p(T_1, T_2)$ and $U = q(U_1, U_2)$. First assume that $p \neq q$. Since $<$ is a total order we have either $p < q$ or $q < p$, by symmetry we may assume $p < q$. Since T and U are in canonical form p does not in occur in any of the trees T_1, T_2 and U. From $T \simeq U$ we then conclude $T_1 \simeq U$ and $T_2 \simeq U$. From the induction hypothesis we conclude $T_1 = U = T_2$, contradicting the assumption that T is in canonical form.

In the remaining case we have $T = p(T_1, T_2)$ and $U = p(U_1, U_2)$. Since T and U are in canonical form p does not in occur in any of the trees T_1, T_2, U_1 and U_2. From $T \simeq U$ we then conclude $T_1 \simeq U_1$ and $T_2 \simeq U_2$. From the induction hypothesis we then conclude $T_1 = U_1$ and $T_2 = U_2$. Hence $T = p(T_1, T_2) = p(U_1, U_2) = U$. □

Theorem 4 *For decision trees T, U we have $T \equiv_{\mathcal{E}} U$ if and only if $T \simeq U$.*

Proof: The 'only if'-part is soundness which follows immediately from the fact that all rules are sound. The 'if'-part (completeness) follows from soundness, Theorem 2 and Lemma 3. □

Combining Theorems 2 and 4 yields a straightforward way to decide whether two decision trees are equivalent or not: reduce them to canonical form and look whether they are syntactically equal. However, it can happen that the canonical form has size exponential in the size of the original decision tree, even if you

may choose a suitable ordering $<$ yourself. Hence worst case this procedure for establishing equivalence is of exponential complexity. A straightforward quadratic procedure for establishing equivalence is well-known; one version is presented in [11].

We conclude this section by an example that indeed the size of a canonical form can be exponential in the size of the original decision tree.

Example 5 Let n be any natural number. Let A consist of $p_1, p_2, \ldots, p_n$, $q_1, q_2, \ldots, q_n$, r and define inductively

$$T_0 = U_0 = \mathsf{false}, \quad T_i = p_i(q_i(\mathsf{true}, \mathsf{false}), T_{i-1}), \quad U_i = q_i(p_i(\mathsf{true}, \mathsf{false}), U_{i-1}),$$

for $i = 1, \ldots, n$, and $V = r(T_n, U_n)$. Clearly V is a decision tree of size $\#V = 4n+1$. In [12] it has been proved that for every order $<$ on A the corresponding canonical form of V has a size exceeding $2^{n/2}$, which is indeed exponential in the size of V.

3 Sharing

A term can be seen as a tree. For measuring the space complexity, the size of a term is usually defined as the number of nodes of this tree. For efficiency reasons, most implementations apply the *sharing* technique. A subterm is stored at a certain location in the memory of the machine, various occurrences of the same subterm are replaced by a pointer to this single location. This shared representation can be seen as a directed acyclic graph (DAG). It is allowed that nodes have more than one parent, but no cycles are introduced by sharing a term.

A natural size of the shared representation is the number of nodes in this DAG. We will consider the *maximally* shared representation of terms, in which each subterm occurs exactly once. In the sequel, with shared, we always mean maximally shared. First we formalize this notion of sharing for genral terms.

A signature Σ is defined to be a set of symbols f each having an arity $\mathsf{ar}(f) \in \mathsf{N}$. We define a shared term over a signature Σ to be a four-tuple $(X, x_0, \mathsf{root}, \mathsf{arg})$, where

- X is a non-empty set;
- $x_0 \in X$;
- $\mathsf{root} : X \to \Sigma$;
- $\mathsf{arg} : X \times \mathsf{N} \to X$ is a partial function for which $\mathsf{arg}(x, i)$ is defined if and only if $1 \leq i \leq \mathsf{ar}(\mathsf{root}(x))$.

Intuitively, x_0 represents the full term, X represents the set of all subterms, $\mathsf{root}(x)$ represents the head symbol of the subterm x, and $\mathsf{arg}(x, i)$ represents the i-th argument of the subterm x. This framework describes both finite and infinite shared terms.

Up to the names of elements of X ordinary terms are transformed canonically to shared terms by the function share. In order to give a definition we need an injective function enum choosing a name $\mathsf{enum}(t)$ for every term t. We define share inductively as follows

$$\mathsf{share}(c) = (\{c\}, c, \mathsf{root}, \mathsf{arg})$$

for every constant c, $\mathsf{root}(c) = c$ and $\mathsf{arg}(c, i)$ is undefined for every i. Assume that for each $i \leq n$, $\mathsf{share}(t_i)$ is inductively defined, and equals $(X_i, x_i, \mathsf{root}_i, \mathsf{arg}_i)$. Define

$$\mathsf{share}(f(t_1, \ldots, t_n)) = (\bigcup_{i=1}^{n} X_i \cup \{x_0\}, x_0, \mathsf{root}, \mathsf{arg}),$$

where

- x_0 is defined to be $\mathsf{enum}(f(t_1, \ldots, t_n))$,
- $\mathsf{root}(x_0) = f$,
- $\mathsf{root}(x) = \mathsf{root}_i(x)$ if $x \in X_i$,
- $\mathsf{arg}(x_0, j) = x_j$,
- $\mathsf{arg}(x, j) = \mathsf{arg}_i(x, j)$ if $x \in X_i$,

for every i, j. Due to injectivity and $\mathsf{root}(\mathsf{enum}(f(t_1, \ldots, t_n))) = f$ and $\mathsf{arg}(\mathsf{enum}(f(t_1, \ldots, t_n)), j) = \mathsf{enum}(t_j)$ we obtain that always $x_0 \notin \bigcup_{i=1}^{n} X_i$ and $\mathsf{root}_i(x) = \mathsf{root}_k(x)$ and $\mathsf{arg}_i(x, j) = \mathsf{arg}_k(x, j)$ for $x \in X_i \cap X_k$, by which root and arg are well-defined.

Since in the first argument of $\mathsf{share}(f(t_1, \ldots, t_n))$ the union is taken, multiple occurrences of subterms are stored only once. The memory needed for storing $\mathsf{share}(t) = (X, x_0, \mathsf{root}, \mathsf{arg})$ is linear in the size of X, representing the number of different subterms. So we define the shared size of a term:

$$\#_{sh}(t) = \#\{s \mid s \text{ is a subterm of } t.\}$$

The shared size of a term can be much smaller than the tree size as is illustrated by the next example. This is exactly the reason that sharing is applied.

Example 6 Define $T_0 = \mathsf{true}$ and $U_0 = \mathsf{false}$. For binary symbols $p_1, p_2, p_3, \ldots$ define inductively $T_n = p_n(T_{n-1}, U_{n-1})$ and $U_n = p_n(U_{n-1}, T_{n-1})$. Considerinf T_n as a term its size $\#(T_n)$ is exponential in n. However, the only subterms of T_n are true, false, and T_i and U_i for $i < n$, hence $\#_{sh}(T_n)$ is linear in n.

The next theorem shows that by sharing no information is lost: by some function unsh the original term can be reconstructed from the shared term. Here unsh is defined inductively as follows:

$$\mathsf{unsh}(X, x_0, \mathsf{root}, \mathsf{arg}) = \mathsf{root}(x_0)(t_1, \ldots, t_n)$$

where $t_j = \mathsf{unsh}(X, \mathsf{arg}(x, j), \mathsf{root}, \mathsf{arg})$ for $j = 1, \ldots, \mathsf{ar}(\mathsf{root}(x_0))$. Note that unsh is not well-defined for all tuples $(X, x_0, \mathsf{root}, \mathsf{arg})$: if a cyclic behaviour is involved then such a tuple represents an infinite term.

Theorem 7 *Let t be any term. Then* $\mathsf{unsh}(\mathsf{share}(t))$ *is well-defined and equals* t.

Proof: The theorem follows from the slightly stronger assertion

> Let $\mathsf{share}(t) = (X, x_0, \mathsf{root}, \mathsf{arg})$. Let $X \subseteq X'$, $\mathsf{root}' : X' \to \Sigma$ and $\mathsf{arg}' : X' \times \mathsf{N} \to X'$ satisfy $\mathsf{root}'(x) = \mathsf{root}(x)$ and $\mathsf{arg}'(x, j) = \mathsf{arg}(x, j)$ for all $x \in X$ and all j. Then $\mathsf{unsh}(X', x_0, \mathsf{root}', \mathsf{arg}')$ is well-defined and equals t.

which is straightforwardly proved by induction on the structure of t. □

Maximal sharing is essentially the same as what is called the *fully collapsed tree* in [10]. In [8] it is shown that the maximally shared representation is unique, and that the original term can be reconstructed from it.

In implementations some care has to be taken in order to keep terms maximally shared. In essence, when constructing a term, a hash table is used to find out whether a node representing this term exists already. If so, this node is reused; otherwise a new node is created. We also refer to the `ATerm` library [3], which is a C-library offering a data type for terms, that are internally stored maximally shared. The main operations are constructing and deconstructing terms in constant time, and unreferenced terms are garbage collected automatically. Furthermore, all BDD-packages can be seen as implementing the idea of maximal sharing.

We now study the time complexity of a term t. In term rewriting this is usually defined as the length of the maximal reduction sequence from t to normal form. Note that all occurrences of the same redex have to be contracted one by one. Because in the shared representation all distinct subterms occur once, it is reasonable to count the contraction of these subterms only once.

Although it is possible to define the rewrite relation on DAGs, this is quite complicated. Note that if a subterm is rewritten, then this should be noticed by all referring nodes. Also note that if $C[D[l^\sigma]]$ reduces to $C[D[r^\sigma]]$, then $D[r^\sigma]$ may occur somewhere else in $C[]$, so after contracting the redex, a number of sharing steps are needed to remove the duplicated nodes from $D[]$.

These problems are partly solved in [1], where a data structure is invented for representing BEDs (a generalization on BDDs). Extra indirections are inserted from nodes to their reduced versions. This technique was already used in [8] on an implementation of rewriting with maximal sharing, called *Unlimp*.

In order to avoid these complexities, we introduce the *shared rewrite relation* on terms. In usual unshared rewriting a rewrite step consists of writing the term as $C[l^\sigma]$ for some context C, some substitution σ and some rewrite rule $l \to r$, and replace the term by $C[r^\sigma]$. In shared rewriting not only this single occurrence of l^σ is replaced by r^σ, but by sharing also every other occurrence of l^σ. By this observation we define shared rewriting without explicitly referring to the shared terms.

Definition 8 *Between two terms t and t' there is a shared rewrite step $t \Rightarrow_R t'$ with respect to a rewrite system R if $t = C[l^\sigma, \ldots, l^\sigma]$ and $t' = C[r^\sigma, \ldots, r^\sigma]$ for*

one rewrite rule $l \to r$ in R, some substitution σ and some multi-hole context C having at least one hole for which l^σ is not a subterm of C.

We will take the maximum number of $\Rightarrow$-steps from t as the time complexity of computing t.

Both in unshared rewrite steps $\to_R$ and shared rewrite steps $\Rightarrow_R$ the subscript R is often omitted if no confusion is caused.

We now study some properties of the rewrite relation $\Rightarrow_R$. The following lemmas are straightforward from the definition.

Lemma 9 *If $t \Rightarrow t'$ then $t \to^+ t'$.*

Lemma 10 *If $t \to t'$ then a term t'' exists satisfying $t' \to^* t''$ and $t \Rightarrow t''$.*

The next theorem shows how the basic rewriting properties are preserved by sharing. In particular, if $\to$ is terminating and all critical pairs converge, then termination and confluence of $\Rightarrow$ can be concluded too.

Theorem 11 *(1) If $\to$ is terminating then $\Rightarrow$ is terminating too.*

(2) A term is a normal form with respect to $\Rightarrow$ if and only if it is a normal form with respect to $\to$.

(3) If $\Rightarrow$ is weakly normalizing and $\to$ has unique normal forms, then $\Rightarrow$ is confluent.

(4) If $\to$ is confluent and terminating then $\Rightarrow$ is confluent and terminating too.

Proof: Part (1) follows directly from Lemma 9.

If t is a normal form with respect to $\to$ then it is a normal form with respect to $\Rightarrow$ by Lemma 9. If t is a normal form with respect to $\Rightarrow$ then it is a normal form with respect to $\to$ by Lemma 10. Hence we have proved part (2).

For part (3) assume $s \Rightarrow^* s_1$ and $s \Rightarrow^* s_2$. Since $\Rightarrow$ is weakly normalizing there are normal forms n_1 and n_2 with respect to $\Rightarrow$ satisfying $s_i \Rightarrow^* n_i$ for $i = 1, 2$. By part (2) n_1 and n_2 are normal forms with respect to $\to$; by Lemma 9 we have $s \to^* n_i$ for $i = 1, 2$. Since $\to$ has unique normal forms we conclude $n_1 = n_2$. Since $s_i \Rightarrow^* n_i$ for $i = 1, 2$ we proved that $\Rightarrow$ is confluent.

Part (4) is immediate from part (1) and part (3). □

Note that Theorem 11 holds for any two abstract reduction systems $\to$ and $\Rightarrow$ satisfying Lemmas 9 and 10 since the proof does not use anything else.

Example 12 (Due to Vincent van Oostrom) Not for all assertions in Theorem 11 the converse holds. For instance, the rewrite system consisting of the two rules $f(0,1) \to f(1,1)$ and $1 \to 0$ admits an infinite reduction $f(0,1) \to f(1,1) \to f(0,1) \to \cdots$, but the shared rewrite relation $\Rightarrow$ is terminating.

For preservation of confuence the combination of termination is essential, as is shown by the rewrite system consisting of the two rules $0 \to f(0,1)$ and $1 \to f(0,1)$. This system is confluent since it is orthogonal, but $\Rightarrow$ is not even locally confluent since $f(0,1)$ reduces to both $f(0, f(0,1))$ and $f(f(0,1),1)$, not having a common $\Rightarrow$-reduct.

Notions on reduction strategies like innermost and outermost rewriting carry over to shared rewriting as follows. As usual a redex is defined to be a subterm of the shape l^σ where $l \to r$ is a rewrite rule and σ is a substitution. A deterministic (one step) reduction strategy is a function that maps every term that is not in normal form to one of its redexes, for instance the leftmost innermost strategy. A non-deterministic reduction strategy is a function that maps every term that is not in normal form to a non-empty set of its redexes, being the redexes that are allowed to be reduced. For instance, in the innermost strategy the set of redexes is chosen for which no proper subterm is a redex itself. This naturally extends to shared rewriting: choose a redex in the set of allowed redexes, and reduce all occurrences of that redex. Note that it can happen that some of these occurrences are not in the set of allowed redexes. For instance, for the two rules $f(x) \to x$, $a \to b$ the shared reduction step $g(a, f(a)) \Rightarrow g(b, f(b))$ is an outermost reduction, while only one of the two occurrences of the redex a is outermost.

4 Reduced OBDDs

Normally a BDD (binary decision diagram) is defined to be a decision tree in which sharing is allowed. An OBDD (ordered binary decision diagram) then is a BDD in which on every path from the root to a leaf the atoms occur only in strictly increasing order, with respect to some fixed total order on the atoms. The main motivation for OBDDs is that there is a natural notion of reduced OBDD (ROBDD) in such a way that it is a unique representation for boolean functions that often can be found reasonably efficiently. Unicity has many strong consequences. For instance, a boolean formula is satisfiable if and only if its ROBDD is not equal to false, and it is a tautology if and only if its ROBDD is equal to true. In our terminology it is very easy to define ROBDDs and prove uniqueness of representation.

Definition 13 *Let $<$ be a total order on A. A ROBDD with respect to $<$ is a decision tree t in canonical form with respect to $<$, in maximally shared representation.*

Usually a ROBDD is defined to be an OBDD in which no node occurs for which the left branch and the right branch point to the same node, and no two nodes labelled by the same symbol occur for which both the two left branches point to the same node and the two right branches point to the same node. This definition coincides with our definition: the first condition due to canonical form, the second due to maximal sharedness.

Theorem 14 *Let $<$ be a total order on A. Then every boolean function can uniquely be represented by a ROBDD with respect to $<$.*

Proof: Every boolean function can be represented by a decision tree. After reducing to canonical form and sharing the desired ROBDD is found. Unicity follows from Lemma 3 and unicity of maximal sharing. □

Next we describe how an arbitrary propositional formula can be transformed to a ROBDD. Just like reducing arbitrary decision trees to canonical form we do this by rewriting. Due to sharing the basic steps of rewriting will be $\Rightarrow$ instead of $\rightarrow$.

One quite simple approach would be to give rewrite rules that first transform the propositional formula to a decision tree and then apply DT until a canonical form has been reached. Although this approach is simple and correct, we do not follow it since it will be very inefficient. Instead we develop an approach by which the standard BDD algorithms based on Bryant's *apply*-function can be mimicked. This *apply*-function computes the ROBDD for $T * U$ for ROBDDs T and U and binary propositional operations $*$ in complexity $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$.

We assume that the propositional formula is constructed from boolean atoms from a set A, the values true and false, the unary operation $\neg$ and binary operations $\vee$, $\wedge$ and xor, all with their usual meaning. Other operations like implication and equivalence can either easily be added to the framework, or alternatively they can be expressed in the other operations without affecting efficiency considerations. The latter is the reason for including xor: generally formulas including equivalence and/or xor can not be reprented in formulas of the same complexity without them.

As a first step every occurrence of an atom p in the formula is replaced by $p(\mathsf{true}, \mathsf{false})$, being the decision tree in canonical form representing the propositional formula p. In this way the formula is represented as a term over the signature consisting of constants true and false, the unary symbol $\neg$ and in which all elements of A and the symbols $\vee$, $\wedge$ and xor are binary symbols. Next we give a rewrite system BDD by which the propositional symbols are propagated through the term and eventually removed, reaching the ROBDD as the normal form. For the binary symbols from A we use prefix notation, for the symbols $\vee$, $\wedge$ and xor we keep the infix notation as is usual in propositional formulas.

The rewrite system BDD is defined to consist of the rules

$$
\begin{array}{rcll}
p(x,x) & \rightarrow & x & \text{for all } p \\
\neg p(x,y) & \rightarrow & p(\neg x, \neg y) & \text{for all } *,\ p \\
p(x,y) * p(z,w) & \rightarrow & p(x * z, y * w) & \text{for all } *,\ p \\
p(x,y) * q(z,w) & \rightarrow & p(x * q(z,w), y * q(z,w)) & \text{for all } *,\ p < q \\
q(x,y) * p(z,w) & \rightarrow & p(q(x,y) * z, q(x,y) * w) & \text{for all } *,\ p < q
\end{array}
$$

$$
\begin{array}{rcl}
\neg\mathsf{true} & \rightarrow & \mathsf{false} \\
\neg\mathsf{false} & \rightarrow & \mathsf{true} \\
\mathsf{true} \vee x & \rightarrow & \mathsf{true} \\
x \vee \mathsf{true} & \rightarrow & \mathsf{true} \\
\mathsf{false} \vee x & \rightarrow & x \\
x \vee \mathsf{false} & \rightarrow & x
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{true} \wedge x & \rightarrow & x \\
x \wedge \mathsf{true} & \rightarrow & x \\
\mathsf{false} \wedge x & \rightarrow & \mathsf{false} \\
x \wedge \mathsf{false} & \rightarrow & \mathsf{false} \\
\mathsf{true}\ \mathsf{xor}\ x & \rightarrow & \neg x \\
x\ \mathsf{xor}\ \mathsf{true} & \rightarrow & \neg x \\
\mathsf{false}\ \mathsf{xor}\ x & \rightarrow & x \\
x\ \mathsf{xor}\ \mathsf{false} & \rightarrow & x
\end{array}
$$

Here p ranges over A and $*$ ranges over the symbols $\vee$, $\wedge$ and xor. The rules of the shape $p(x,x) \rightarrow x$ are called *idempotence rules*, all other rules are called *essential rules*.

We have defined *BDD* in such a way that terms are only rewritten to logically equivalent terms. Hence if some term rewrites in some way by *BDD* to a ROBDD, we may conclude that this reduced OBDD is the unique representation for the original term.

The rewrite system *BDD* is terminating since every left hand side is greater than the corresponding right hand side with respect to any recursive path order for a precedence $\succ$ satisfying $\mathsf{xor} \succ \neg$ and $* \succ p$ for $* \in \{\neg, \vee, \wedge, \mathsf{xor}\}$ and $p \in A$. Hence reducing will lead to a normal form, and it is easily seen that ground normal forms do not contain symbols $\neg, \vee, \wedge, \mathsf{xor}$.

The rewrite system *BDD* is not confluent, for instance if $p > q$ the term $p(q(\mathsf{false}, \mathsf{true}), q(\mathsf{false}, \mathsf{true})) \wedge p(\mathsf{false}, \mathsf{true})$ reduces to the two distinct normal forms $p(\mathsf{false}, q(\mathsf{false}, \mathsf{true}))$ and $q(\mathsf{false}, p(\mathsf{false}, \mathsf{true}))$. Moreover, we see that *BDD* admits ground normal forms that are not in canonical form. However, when starting with a propositional formula this can not happen due to the following invariant:

> **Invariant:**
>
> For every subterm of the shape $p(T, U)$ for $p \in A$ all symbols $q \in A$ occurring in T or U satisfy $p < q$.

In a propositional formula in which only every atom p is replaced by $p(\mathsf{true}, \mathsf{false})$ this clearly holds since $T = \mathsf{true}$ and $U = \mathsf{false}$ for every subterm of the shape $p(T, U)$. Further for all rules of *BDD* it is easily checked that if the invariant holds for some term, after application of an *BDD*-rule it remains to hold. Hence for normal forms of propositional formulas the invariant holds. Due to the idempotence rules we now conclude that these normal forms are in canonical form. We have proved the following theorem.

Theorem 15 *Let Φ be a propositional formula over A. Replace every atom $p \in A$ occurring in Φ by $p(\mathsf{true}, \mathsf{false})$ and reduce the resulting term to normal form with respect to $\Rightarrow_{BDD}$. Then the resulting normal form is the unique ROBDD of Φ.*

In this way we have described the process of constructing the unique ROBDD purely by rewriting. Instead of having a deterministic algorithm for this construction as described in the literature [4, 9], we still have a lot of freedom in choosing the strategy for reducing to normal form, but one strategy may be much more efficient than another. We will show that the leftmost innermost strategy, even when adapted to shared rewriting, may be extremely inefficient, but we will also show that the standard algorithm from the literature can essentially be mimicked by a layerwise reduction strategy having the same complexity. Moreover, we introduce a lazy strategy that is essentially better for some examples than the standard algorithm.

Example 16 As in Example 6 define $T_0 = \mathsf{true}$ and $U_0 = \mathsf{false}$, and define inductively $T_n = p_n(T_{n-1}, U_{n-1})$ and $U_n = p_n(U_{n-1}, T_{n-1})$.

Both T_n and U_n are in canonical form, hence can be considered as ROBDDs. Both are the ROBDDs of simple propositional formulas, in particular for odd

n the term T_n is the ROBDD of $\mathsf{xor}_{i=1}^{n}\ p_i$ and U_n of $\neg(\mathsf{xor}_{i=1}^{n}\ p_i)$, and for even n the other way around. In fact they describe the *parity* functions yielding true if and only if the number of i-s for which p_i holds is even, or odd.

Surprisingly, for every n both for $\neg(T_n)$ and $\neg(U_n)$ $\Rightarrow_{BDD}$-reduction to normal form by the leftmost-innermost strategy requires $2^n - 1$ $\neg$-steps, where a $\neg$-step is defined to be an application of a rule $\neg p(x,y) \to p(\neg x, \neg y)$. We prove this by induction on n. For $n = 0$ it trivially holds. For $n > 0$ the first reduction step is

$$\neg(T_n) \Rightarrow_{BDD} p_n(\neg(T_{n-1}), \neg(U_{n-1})).$$

The leftmost-innermost reduction continues by reducing $\neg(T_{n-1})$. During this reduction no $\neg$-redex is shared in $\neg(U_{n-1})$ since $\neg(U_{n-1})$ contains only one $\neg$-symbol that is too high in the tree. Hence $\neg(T_{n-1})$ is reduced to normal form with $2^{n-1} - 1$ $\neg$-steps due to the induction hypothesis, without affecting the right part $\neg(U_{n-1})$ of the term. After that another $2^{n-1} - 1$ $\neg$-steps are required to reduce $\neg(U_{n-1})$, making the total of $2^n - 1$ $\neg$-steps. For $\neg(U_n)$ the argument is similar, concluding the proof.

Although the terms encountered in this reduction are very small in the shared representation, we see that by this strategy every $\Rightarrow$-step consists of one single $\to$-step, of which exponentially many are required.

We say that a subterm V of a term T is an *essential redex* if $V = l^\sigma$ for some substitution σ and some essential rule $l \to r$ in *BDD*.

Proposition 17 *Let T, U be ROBDDs.*

- *If $\neg T \Rightarrow^*_{BDD} V$ then every essential redex in V is of the shape $\neg T'$ for some subterm T' of T.*
- *If $T * U \Rightarrow^*_{BDD} V$ for $* = \vee$ or $* = \wedge$ then every essential redex in V is of the shape $T' * U'$ for some subterm T' of T and some subterm U' of U.*
- *If T xor $U \Rightarrow^*_{BDD} V$ then every essential redex in V is of the shape T' xor U' or $\neg T'$ or $\neg U'$ for some subterm T' of T and some subterm U' of U.*

Proof: This proposition immediately follows from its unshared version: let T, U be decision trees in canonical form and replace $\Rightarrow_{BDD}$ in all three assertions by $\to_{BDD}$. This unshared version is proved by induction on the reduction length of $\to^*_{BDD}$ and considering the shape of the rules of *BDD*. □

The problem in the exponential leftmost innermost reduction above is that during the reduction very often the same redex is reduced. The key idea now is that in a *layerwise* reduction every essential redex is reduced at most once.

Definition 18 *An essential redex l^σ is called a p-redex for $p \in A$ if p is the smallest symbol occurring in l^σ with respect to $<$. An essential redex l^σ is called an ∞-redex if no symbol $p \in A$ occurs in l^σ; define $p < \infty$ for all $p \in A$.*

A redex is called layerwise if either

- *it is a redex with respect to an idempotence rule, or*
- *it is a p-redex for $p \in A \cup \{\infty\}$, and no q-redex exists for $q < p$ exists, and if the root of the redex is $\neg$ then no p-redex exists of which the root is* xor.

$A \Rightarrow_{BDD}$-reduction is called layerwise if every step consists of the reduction of all occurrences of a layerwise redex.

Note that every essential redex is a p-redex for some $p \in A \cup \{\infty\}$.

Clearly every term not in normal form contains a layerwise redex, hence layerwise reduction always leads to the unique normal form. Just like innermost and outermost reduction, layerwise reduction is a non-deterministic reduction strategy. We will show that layerwise reduction leads to normal forms efficiently for suitable terms, due to the following proposition.

Proposition 19 *Let T, U be ROBDDs. In every layerwise $\Rightarrow_{BDD}$-reduction of $\neg T$, $T \vee U$, $T \wedge U$ or T xor U every essential redex is reduced at most once.*

Proof: Assume that an essential redex l^σ is reduced twice:

$$C[l^\sigma] \Rightarrow^+ C'[l^\sigma] \Rightarrow \cdots$$

As already remarked l^σ is a p-redex for some $p \in A \cup \{\infty\}$. Since the reduction is layerwise every reduction step is either an idempotence step or a reduction of a p-redex for this particular p. Due to Proposition 17 and the shape of the rules the only kind of new p-redexes that can be created in this reduction is a p-redex having $\neg$ as its root, obtained by reducing a p-redex having xor as its root. So this p-redex with root xor already occurs in $C[l^\sigma]$. Since the reduction is layerwise the root of l^σ is not $\neg$. We conclude that the p-redex l^σ in $C'[l^\sigma]$ is not created during this reduction, hence it already occurred in the first term $C[l^\sigma]$. Since we apply shared rewriting this occurence of l^σ was already reduced in the first step, contradiction. □

Theorem 20 *Let T be a ROBDD. Then every layerwise $\Rightarrow_{BDD}$-reduction of $\neg T$ contains at most $\#_{sh}(T)$ steps.*

Let T, U be ROBDDs. Then every layerwise $\Rightarrow_{BDD}$-reduction of $T \vee U$, $T \wedge U$ or T xor U contains $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$ steps.

Proof: If a layerwise reduction of $\neg T$ contains an idempotence step $V \Rightarrow V'$, then this idempotence step was also possible on the the original term T, contradicting the assumption that T is a ROBDD. Hence a layerwise reduction of $\neg T$ consists only of reductions of essential redexes, and by Proposition 17 the number of candidates is at most $\#_{sh}(T)$. By Proposition 19 each of these possible essential redexes is reduced at most once, hence the total number of steps is at most $\#_{sh}(T)$.

Let V be either $T \vee U$, $T \wedge U$ or T xor U. Then a layerwise reduction of V consists of a combination of reductions of essential redexes and a number

of idempotence steps. By Proposition 17 the number of candidates for essential redexes is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, each of which is reduced at most once by Proposition 19. Hence the total number of reductions of essential redexes is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. Since in every reduction of an essential redex the shared size $\#_{sh}$ increases by at most one, and by every idempotence step $\#_{sh}$ decreases by at least one, the total number of idempotence steps is at most $\#_{sh}(V) + \mathcal{O}(\#_{sh}(T) * \#_{sh}(U)) = \mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. Hence the total number of steps is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. □

Write $\mathsf{apply}(T)$ for layerwise reducing a term T to normal form; due to Theorem 20 apply can be used as an alternative for *apply* as mentioned before, with comparable efficiency. We now can define an algorithm reduce to find the ROBDD for a propositional formula Φ:

$$\begin{aligned}
\mathsf{reduce}(\mathsf{true}) &= \mathsf{true} \\
\mathsf{reduce}(\mathsf{false}) &= \mathsf{false} \\
\mathsf{reduce}(p) &= p(\mathsf{true}, \mathsf{false}) \\
\mathsf{reduce}(\Phi \vee \Psi) &= \mathsf{apply}(\mathsf{reduce}(\Phi) \vee \mathsf{reduce}(\Psi)) \\
\mathsf{reduce}(\Phi \wedge \Psi) &= \mathsf{apply}(\mathsf{reduce}(\Phi) \wedge \mathsf{reduce}(\Psi)) \\
\mathsf{reduce}(\Phi \ \mathsf{xor}\ \Psi) &= \mathsf{apply}(\mathsf{reduce}(\Phi) \ \mathsf{xor}\ \mathsf{reduce}(\Psi))
\end{aligned}$$

Roughly speaking this function reduce mimics the usual algorithm as described in the literature.

The procedure above can be viewed as a particular strategy on shared terms. In terms containing more than one boolean connective, one of the innermost connectives is pushed down completely, using layerwise reductions. In fact, this is the *up-all* algorithm, invented by [1].

However, other strategies are also conceivable. For instance, we could device a strategy which brings the smallest variable to the root very quickly. To this end, we define *head normal forms* to be terms of the form false, true and $p(T, U)$. The *lazy strategy* forbids reductions inside T in subterms of the form $T*U$, $U*T$ and $\neg T$, in case T is in head normal form.

Lemma 21 *Each (unshared) lazy reduction sequence from T, leads to a head normal form in at most $2\#(T)$ reductions.*

Proof: Induction on T. The cases false, true and $p(T, U)$ are trivial.

Let $T = P * Q$, with $* \in \{\mathsf{xor}, \wedge, \vee\}$: Let $\#(P) = m$ and $\#(Q) = n$. By induction hypothesis, P reduces to head normal form in at most $2m$ steps. So the lazy strategy allows at most $2m$ reductions in the left hand side of $P * Q$. Similarly, in the right hand side at most $2n$ steps are admitted.

Hence after at most $2(m+n)$ steps, $*(P, Q)$ is reduced to one of: $p(P_1, P_2) * q(Q_1, Q_2)$ or $b * Q_1$ or $P_1 * b$, where $b \in \{\mathsf{false}, \mathsf{true}\}$ and P_i and Q_i are in head normal form for $i = 1, 2$. In most of the cases this reduces to head normal form in the next step, for true xor Q_1 and P_1 xor true it takes two steps to reach a head normal form. So we use at most $2(m + n) + 2 = 2\#(T)$ steps.

Case $T = \neg P$ is similar but easier. □

Example 22 Let Φ be a formula of size m, whose ROBDD-representation is exponentially large in m. Assume that variable p is smaller than all variables occurring in formula Φ. Consider the formula $p \wedge (\Phi \wedge \neg p)$, which is clearly unsatisfiable. Note that the traditional algorithm using *apply* will as an intermediate step always completely build the ROBDD for Φ, which is known to be exponential.

We now show that the lazy strategy has linear time complexity. Replace each prositional variable q by $q(\mathsf{true}, \mathsf{false})$, transforming Φ to Φ'. Using the lazy reduction strategy sketched above, we get a reduction of the following shape:

$$\begin{array}{ll} & p(\mathsf{true}, \mathsf{false}) \wedge (\Phi' \wedge \neg p(\mathsf{true}, \mathsf{false})) \\ \to^{n+1} & p(\mathsf{true}, \mathsf{false}) \wedge (q(\Phi_1, \Phi_2) \wedge p(\neg\mathsf{true}, \neg\mathsf{false})) \\ \to & p(\mathsf{true}, \mathsf{false}) \wedge p(q(\Phi_1, \Phi_2) \wedge \neg\mathsf{true}, q(\Phi_1, \Phi_2) \wedge \neg\mathsf{false}) \\ \to & p(\mathsf{true} \wedge (q(\Phi_1, \Phi_2) \wedge \neg\mathsf{true}), \mathsf{false} \wedge (q(\Phi_1, \Phi_2) \wedge \neg\mathsf{false})) \\ \to^{k} & p(\mathsf{false}, \mathsf{false}) \\ \to & \mathsf{false} \end{array}$$

where n is the number of steps applied on Φ' until a head normal form $q(\Phi_1, \Phi_2)$ is reached. This shape is completely forced by the lazy strategy; within the $n+1$ and k steps some non-determinism is present, but always $k \leq 6$. Note that reductions inside Φ_1 and Φ_2 are never permitted. By Lemma 21 we have $n \leq 2m$, so the length of the reduction is linear in m.

Note that we only considered unshared rewriting. In shared rewriting however essentially the same lazy reduction is forced.

The lazy reduction is similar to the *up-one* algorithm in [1]. In [1] an example is shown where *up-one* is relatively efficient, but there additional rewrite rules are used, e.g. x xor $x \to \mathsf{false}$.

5 Conclusions

As achievements of this paper, we summarize:

- A re-development of the basic BDD theory using standard rewriting techniques.
- A simple and elegant abstraction of maximally shared rewriting, and a proof that the transition to shared rewriting preserves completeness (i.e. termination and confluence).
- A TRS *BDD* to construct ROBDDs, generalizing the traditional *apply*-algorithm.
- A layerwise strategy for *BDD*, with provably the same time complexity as the traditional algorithm.
- A lazy strategy for *BDD*, with an example in which this performs much better than the traditional algorithm.

Acknowledgement

We want to thank Vincent van Oostrom for his contribution to the theory of sharing and for many fruitful discussions.

References

[1] Andersen, H. R., and Hulgaard, H. Boolean expression diagrams. In *Twelfth Annual IEEE Symposium on Logic in Computer Science* (Warsaw, Poland, 1997), IEEE Computer Society, pp. 88–98.

[2] Baader, F., and Nipkow, T. *Term Rewriting and All That.* Cambridge University Press, 1998.

[3] Brand, M. v. d., Jong, H. d., Klint, P., and Olivier, P. Efficient annotated terms. *Software Practice en Experience* ((accepted)). See `http://www.wins.uva.nl/~olivierp/aterm.html`.

[4] Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers C-35*, 8 (1986), 677–691.

[5] Bryant, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys 24*, 3 (1992), 293–318.

[6] Burch, J., Clarke, E., Long, D., McMillan, K., and Dill, D. Symbolic model checking for sequential circuit verification. *IEEE Trans. Computer Aided Design 13*, 4 (1994), 401–424.

[7] Clarke, E., Emerson, E., and Sistla, A. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems 8*, 2 (1986), 244–263.

[8] Kahrs, S. Unlimp: Uniqueness as a leitmotiv for implementation. In *Proc. Programming Language Implementation and Logic Programming* (1992), vol. 631 of *Lecture Notes in Computer Science*, Springer, pp. 115–129.

[9] Meinel, C., and Theobald, T. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications.* Springer, 1998.

[10] Plump, D. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages* (1999), H.-J. K. H. Ehrig, G. Engels and G. Rozenberg, Eds., World Scientific, pp. 3–61.

[11] Zantema, H. Decision trees: Equivalence and propositional operations. In *Proceedings 10th Netherlands/Belgium Conference on Artificial Intelligence (NAIC'98)* (November 1998), H. L. Poutré and J. van den Herik, Eds., pp. 157 – 166. Extended version appeared as report UU-CS-1998-14, Utrecht University.

[12] Zantema, H., and Bodlaender, H. L. Sizes of decision tables and decision trees. Tech. Rep. UU-CS-1999-31, Utrecht University, Department of Computer Science, 1999. Available via `http://www.cs.uu.nl/docs/research/publication/TechRep.html`.