Binary Decision Diagrams by Shared Rewriting

Jaco van de Pol[1]* and Hans Zantema[1,2]**

[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. In this paper we propose a uniform description of basic BDD theory and algorithms by means of term rewriting. Since a BDD is a DAG instead of a tree we need a notion of shared rewriting and develop appropriate theory. A rewriting system is presented by which canonical forms can be obtained. Various reduction strategies give rise to different algorithms. A *layerwise* strategy is proposed having the same time complexity as the traditional *apply*-algorithm, and the *lazy* strategy is studied, which resembles the existing *up-one*-algorithm. We show that these algorithms have incomparable performance.

1 Introduction

Equivalence checking and satisfiability testing of propositional formulas are basic but hard problems in many applications, including hardware verification [4] and symbolic model checking [5]. Binary decision diagrams (BDDs) [2,3,8] 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 are trivial. Constructing the canonical form however, can be exponential.

Various extensions to the basic data-type have been proposed, like DDDs [9], BEDs [1] and EQ-BDDs [6]. Many variants of Bryant's original *apply*-algorithm for computing boolean combinations of ROBDDs have been proposed in the literature. Usually, such adaptations are motivated by particular benchmarks, that show a speed-up for certain cases. In many cases, the relative complexity between the variants is not clear and difficult to establish due to the variety of data-types.

Therefore, we propose to use term rewriting systems (TRS) as a uniform model for the study of operations on BDDs. By enriching the signature, extended data types can be modeled. Various different algorithms can be obtained from a fixed TRS by choosing a reduction *strategy*. In our view, this opens the way in which the BDD-world can benefit from the huge amount of research on rewriting strategies (see [7] for an overview).

* Email: `Jaco.van.de.Pol@cwi.nl`
** Email: `hansz@cs.uu.nl`

A complication is that the relative efficiency of BDDs hinges on the maximally shared representation. In Section 2 we present an elegant abstraction of maximally shared graph rewriting, in order to avoid its intricacies. 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 terminating and confluent, then the shared version is so too. This enables us to lift rewrite results from standard term rewriting to the shared setting for free.

In Section 3, we present a TRS for applying logical operations to ROBDDs and prove its correctness. Because a TRS-computation is non-deterministic, this proves the correctness of a whole class of algorithms. In particular, we reconstruct the traditional *apply*-algorithm as an application of the so-called *layerwise* strategy. We also investigate the well-known *innermost* and *lazy* strategies. The lazy strategy happens to coincide with the the *up-one* algorithm in [1] (those authors argue that their *up-all* algorithm is similar to the traditional *apply*).

Finally we provide series of examples to show that the innermost strategy performs quite bad, and that the *apply*-algorithm and the lazy strategy have incomparable complexity. In [1] an example is given for one direction, but this depends on additional structural rules. An extended version of this paper appeared as [11].

2 Shared Term Rewriting

We assume familiarity with standard notions from term rewriting. See [7] for an introduction. The size of a term T is usually measured as the number of its internal nodes, viewed as a *tree*. This is inductively defined as $\#(T) = 0$ if T is a constant or a variable, and $\#(f(T_1, \ldots, T_n)) = 1 + \#(T_1) + \cdots + \#(T_n)$.

However, for efficiency reasons, most implementations apply the *sharing* technique. Each 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). Mathematically, we define the *maximally* shared representation of a term as the set of its subterms. It is clear that there is a one-to-one correspondence between a tree and its maximally shared representation.

A natural size of the shared representation is the number of nodes in the DAG. So we define the shared size of a term:

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

The size of the shared representation can be much smaller than the tree size as illustrated by the next example, which is exactly the reason that sharing is applied.

Example 1. 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 $U_n = p_n(U_{n-1}, T_{n-1})$. Considering 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. □

Maximal sharing is essentially the same as what is called the *fully collapsed tree* in [10]. In implementations some care has to be taken in order to keep terms maximally shared. In essence, when constructing or modifying 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. In order to avoid these difficulties in complexity analysis, we introduce the *shared rewrite relation* $\Rightarrow$ on terms. In a shared rewrite step, all occurrences of a redex have to be rewritten at once. We will take the maximum number of $\Rightarrow$-steps from t as the time complexity of computing t.

Definition 1. *For 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, and such that l^σ is not a subterm of C.*

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

Lemma 2. *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 1. *(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 1.

If t is a normal form with respect to $\to$ then it is a normal form with respect to $\Rightarrow$ by Lemma 1. If t is a normal form with respect to $\Rightarrow$ then it is a normal form with respect to $\to$ by Lemma 2. 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 1 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 1 holds for any two abstract reduction systems $\to$ and $\Rightarrow$ satisfying Lemmas 1 and 2 since the proof does not use anything else.

Example 2. (Due to Vincent van Oostrom) The converse of Theorem 1.1 doesn't hold. 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 confluence the combination with 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. $\square$

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 (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.

3 ROBDD Algorithms as Reduction Strategies

We consider a set A of binary *atoms*, whose typical elements are denoted by $p, q, r, \ldots$. 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.

Given an instance $s : A \to \{\mathsf{true}, \mathsf{false}\}$, every decision tree can be evaluated to either true or false, by interpreting $p(T, U)$ as "if $s(p)$ then T else U". So a decision tree represents a boolean function. 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. Two decision trees T and U are called *equivalent* if they represent the same boolean function.

A decision tree is said to be in *canonical form* with respect to some total 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. A BDD (binary decision diagram) is defined to be a decision tree in which sharing is allowed. An ROBDD (reduced ordered binary decision diagram) can now simply be defined as the maximally shared representation of a decision tree in canonical form.

Theorem 2 (Bryant [2]). *Let $<$ be a total order on A. Then every boolean function can uniquely be represented by an ROBDD with respect to $<$.*

We refer to [11] for our proof of this fact using standard rewriting analysis based on weak normalization and confluence of an appropriate rewrite system, whose normal forms are canonical.

Theorem 2 suggests a way to decide whether two logical formulas are equivalent: bring both expressions to ROBDD form and look whether the results are syntactically equal. We now describe how an arbitrary propositional formula can be transformed to an ROBDD by rewriting. Due to sharing the basic steps of rewriting will be $\Rightarrow$ instead of $\rightarrow$.

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. The signature of the TRS consists of the constants true and false, the unary symbol $\neg$, binary symbols for all elements of A and the binary symbols $\vee$, $\wedge$ and xor, written infix as usually.

Next we give a rewrite system $\mathcal{B}$ by which the propositional symbols are propagated through the term and eventually removed, reaching the ROBDD as the normal form. In Figure 1, p ranges over A and $\diamond$ 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*.

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

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

Fig. 1. The rewrite system $\mathcal{B}$.

We have defined $\mathcal{B}$ in such a way that terms are only rewritten to logically equivalent terms. Hence if a term rewrites in some way by $\mathcal{B}$ to an ROBDD, we may conclude that the result is the unique ROBDD equivalent to the original term (independent of whether the system is confluent).

The rewrite system $\mathcal{B}$ 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 $\mathsf{xor} \succ \neg \succ b$ and $\diamond \succ p$ for $\diamond \in \{\neg, \vee, \wedge, \mathsf{xor}\}$ and $b \in \{\mathsf{false}, \mathsf{true}\}$ 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 $\neg, \vee, \wedge, \mathsf{xor}$. By Theorem 1.(1) this also holds for shared rewriting.

The rewrite system $\mathcal{B}$ is not (ground) confluent, for instance if $q > p$ the term $q(p(\mathsf{false}, \mathsf{true}), p(\mathsf{false}, \mathsf{true})) \wedge q(\mathsf{false}, \mathsf{true})$ 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 $\mathcal{B}$ admits ground normal forms that are not in canonical form. However, when starting with a propositional formula this cannot happen due to the following

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 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 $\mathcal{B}$ it is easily checked that if the invariant holds for some term, after application of a $\mathcal{B}$-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 3. *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 $\Rightarrow_{\mathcal{B}}$. Then the resulting normal form is the ROBDD of Φ.*

In this way we have described the process of constructing the unique ROBDD purely by rewriting. Of course this system is inspired by [2, 8], but instead of having a deterministic algorithm, we now 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 first show that the leftmost innermost strategy, even when adapted to shared rewriting, may be extremely inefficient.

Example 3. As in Example 1 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, respectively.

Surprisingly, for every n both for $\neg(T_n)$ and $\neg(U_n)$ $\Rightarrow_{\mathcal{B}}$-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

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

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 $\rightarrow$-step, of which exponentially many are required. $\square$

We will now show that the standard algorithm based on Bryant's *apply* can essentially be mimicked by a layerwise reduction strategy, having the same complexity. 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 $l \rightarrow r$ in $\mathcal{B}$.

Proposition 1. *Let T, U be ROBDDs.*

- *If $\neg T \Rightarrow^*_{\mathcal{B}} V$ then every essential redex in V is of the shape $\neg T'$ for some subterm T' of T.*
- *If $T \diamond U \Rightarrow^*_{\mathcal{B}} V$ for $\diamond = \vee$ or $\diamond = \wedge$ then every essential redex in V is of the shape $T' \diamond U'$ for some subterm T' of T and some subterm U' of U.*
- *If $T \;\mathsf{xor}\; U \Rightarrow^*_{\mathcal{B}} V$ then every essential redex in V is of the shape $T' \;\mathsf{xor}\; U'$ 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 $\Rightarrow_{\mathcal{B}}$ in all three assertions by $\rightarrow_{\mathcal{B}}$. This unshared version is proved by induction on the reduction length of $\rightarrow^*_{\mathcal{B}}$ and considering the shape of the rules of $\mathcal{B}$. $\square$

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 2. *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 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 $\Rightarrow_{\mathcal{B}}$-reduction is called layerwise if every step consists of the reduction of all occurrences of a layerwise redex.

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 2. *Let T, U be ROBDDs. In every layerwise $\Rightarrow_{\mathcal{B}}$-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:

$$C[l^\sigma] \Rightarrow_{\mathcal{B}}^{+} C'[l^\sigma] \Rightarrow_{\mathcal{B}} \cdots$$

Note that l^σ is a p-redex for some $p \in A \cup \{\infty\}$, because it is essential. 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 1 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 occurrence of l^σ was already reduced in the first step, contradiction. $\square$

Theorem 4. *Let T be an ROBDD. Then every layerwise $\Rightarrow_{\mathcal{B}}$-reduction of $\neg T$ contains at most $\#_{sh}(T)$ steps.*

Let T, U be ROBDDs. Then every layerwise $\Rightarrow_{\mathcal{B}}$-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 $V \Rightarrow_{\mathcal{B}} V'$, then this idempotence step was also possible on the original term T, contradicting the assumption that T is an ROBDD. Hence a layerwise reduction of $\neg T$ consists only of reductions of essential redexes, and by Proposition 1 the number of candidates is at most $\#_{sh}(T)$. By Proposition 2 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 1 the number of candidates for essential redexes is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, each of which is reduced at most once by Proposition 2. 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 $\#_{sh}(V) + \mathcal{O}(\#_{sh}(T) * \#_{sh}(U)) = \mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. So the total number of steps is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. $\square$

The procedure sketched above mimics Bryant's original *apply*-function. On formulas with more than one connective, it is repeatedly applied to one of the innermost connectives, thus removing all connectives step by step. It can also be seen as lifting all propositional atoms, for which reason it is called *up-all* in [1]. Note that this is *not* the same as applying the layerwise strategy on the formula itself.

However, other strategies are also conceivable. For instance, we could device a strategy which brings the smallest atom 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* is defined to forbid reductions inside T in subterms of the form $T \diamond U$, $U \diamond T$ and $\neg T$ in case T is in head normal form. We will show that the lazy strategy is not comparable to the *apply*-algorithm.

Lemma 3. *Each (unshared) lazy reduction sequence from T leads to a head normal form in at most $2\#(T)$ reduction steps.*

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

Let $T = P \diamond Q$, with $\diamond \in \{\mathsf{xor}, \wedge, \vee\}$: 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 $P \diamond Q$. Similarly, in the right hand side at most $2n$ steps are admitted.

Hence after at most $2(m + n)$ steps, $P \diamond Q$ is reduced to one of: $p(P_1, P_2) \diamond q(Q_1, Q_2)$ or $b \diamond Q_1$ or $P_1 \diamond b$, 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 4. Let Φ be a formula of size m, whose ROBDD-representation is exponentially large in m (for instance $\bigvee_{i=1}^{n}(p_i \wedge q_i)$ with $p_i < q_j$ for all i and j [3]). Assume that atom p is smaller than all atoms 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 exponential by assumption.

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

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

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 3 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.

Conversely, it can be proved that for $(\cdots((p_1\ \mathsf{xor}\ p_2)\ \mathsf{xor}\ p_3)\cdots)\ \mathsf{xor}\ p_n$ the *apply*-algorithm determines the ROBDD in time quadratic in n, while the lazy strategy admits reductions of length exponential in n. The proof is similar to that of Example 3 □

The lazy reduction appears to be similar to the *up-one* algorithm in [1]. There it is shown that for certain benchmarks *up-one* is relatively efficient, but there additional rewrite rules are used, e.g. $x \mathsf{\ xor\ } x \rightarrow \mathsf{false}$. We have proved that it can also be an improvement without adding more rules. On the other hand, we gave an example on which the traditional *apply*-algorithm turned out to be better.

4 Conclusion

The TRS approach is promising, as it concisely and flexibly describes the BDD data structure and operations. Extensions to the data structure, like complemented edges, DDDs, BEDs and EQ-BDDs can be obtained basically by extending the signature. Various known algorithms are obtained as different reduction strategies. In this way the relative complexity of various proposals can be analyzed.

Acknowledgment. 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. Bryant, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers C-35*, 8 (1986), 677–691.
3. Bryant, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys 24*, 3 (1992), 293–318.
4. 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.
5. 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.
6. Groote, J., and van de Pol, J. Equational binary decision diagrams. Tech. rep. SEN-R0006, CWI, Amsterdam, 2000. Available via `http://www.cwi.nl/~vdpol/papers/eqbdds.ps.Z`.
7. Klop, J. W. Term rewriting systems. In *Handbook of Logic in Computer Science*, D. G. S. Abramski and T. Maibaum, Eds., vol. 2. Oxford University Press, 1992.
8. Meinel, C., and Theobald, T. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.
9. Møller, J., Lichtenberg, J., Andersen, H. R., and Hulgaard, H. Difference decision diagrams. In *Computer Science Logic* (Denmark, Sept. 1999).
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. van de Pol, J. C., and Zantema, H. Binary decision diagrams by shared rewriting. Tech. Rep. UU-CS-2000-06, Utrecht University, 2000. Also published as CWI report SEN-R0001, Amsterdam. Available via `http://www.cs.uu.nl/docs/research/publication/TechRep.html`.