ELSEVIER

The Journal of Logic and Algebraic Programming 49 (2001) 61–86

THE JOURNAL OF LOGIC AND ALGEBRAIC PROGRAMMING

www.elsevier.com/locate/jlap

A rewriting approach to binary decision diagrams

Hans Zantema [a,*], Jaco van de Pol [b]

[a] *Department of Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands*

[b] *Department of Software Engineering, Centrum voor Wiskunde en Informatica, P.O. Box 94.079, 1090 GB Amsterdam, The Netherlands*

Received 11 January 2001; received in revised form 27 July 2001; accepted 30 July 2001

Abstract

Binary decision diagrams (BDDs) provide an established technique for propositional formula manipulation. In this paper, we present the basic BDD theory by means of standard rewriting techniques. 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 reduced ordered BDDs (ROBDDs) can be obtained and for which uniqueness of ROBDD representation is proved. Next, an alternative rewriting system is presented, suitable for actually computing ROBDDs from formulas. For this rewriting system a *layerwise* strategy is defined, and it is proved that when replacing the classical *apply*-algorithm by layerwise rewriting, roughly the same complexity bound is reached as in the classical algorithm. Moreover, a *layerwise innermost* strategy is defined and it is proved that the full classical algorithm for computing ROBDDs can be replaced by layerwise innermost rewriting without essentially affecting the complexity. Finally a *lazy* strategy is proposed sometimes performing much better than the traditional algorithm.

Keywords: Decision trees; Binary decision diagrams; Term rewriting; Shared rewriting; Reduction strategies; Reduction length

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) [4,5,12,16], 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.

*Corresponding author.

E-mail addresses: h.zantema@tue.nl (H. Zantema), jaco.van.de.pol@cwi.nl (J. van de Pol).

1567-8326/01/$ - see front matter
PII: S1567-8326(01)00013-3

A main goal of the BDD approach is to keep constructing these canonical forms tractable for as many boolean formulas as possible.

Various extensions to the basic data-type have been proposed such as DDDs [13], BEDs [1] and EQ-BDDs [8]. 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.

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 research on rewriting strategies, see [11] for an overview. The second motivation for a rewriting approach to BDD is an educational one. As term rewriting becomes more and more standard (see e.g. the textbook [2]), it is helpful to present the BDD-theory in the standard notation and theory of term rewriting. Finally, in the BDD-world various extensions are emerging, both with respect to the data structure as well as the algorithmics (see e.g. [1,9]). Term rewriting can present a general framework for describing 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 similar time complexity. More precisely, we prove that the number of rewrite steps involved is bounded by the sharp complexity bound of the traditional *apply*-algorithm. In this approach, the *apply*-algorithm is replaced by rewriting, being a basic part of the full traditional algorithm. The next step is to give a rewriting approach for the full algorithm: we define a *layerwise innermost*

reduction strategy for which we show that if the ROBDD is computed by applying the TRS using this strategy, then the required number of rewrite steps does not exceed the known complexity bound of the standard algorithm.

As an alternative strategy the *lazy* strategy is presented. It is proved that by lazy rewriting a topmost symbol is computed in time linear to the shared size of the input term. We also give an example, where lazy rewriting performs much better than any innermost strategy, including the traditional algorithm based on *apply*.

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, the results of some experiments are presented.

This paper grew out of earlier work [15]. In the present paper, we extend the more restrictive definition of layerwise as introduced there, while being able to prove the same main theorem. The current upper bound on head reductions is sharper than in that version, by using the shared size of a term. Moreover, the results for layerwise innermost rewriting and the experiments are new.

2. Decision trees

We consider a finite set A of binary *atoms*, whose typical elements are denoted by $p, q, r, \ldots$ A *valuation* σ over A is defined to be a map from A to {true, false}; intuitively for an atom p and a valuation σ the value $\sigma(p)$ represents whether the boolean atom p holds for the valuation σ 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. More precisely, we define a decision tree over A is 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 $[\![T]\!]_\sigma$ can be assigned to every decision tree T and every valuation σ, inductively defined as follows:

$$\begin{aligned}
[\![\mathsf{true}]\!]_\sigma &= \mathsf{true}\\
[\![\mathsf{false}]\!]_\sigma &= \mathsf{false}\\
[\![p(T,U)]\!]_\sigma &= [\![T]\!]_\sigma \quad \text{if } \sigma(p) = \mathsf{true}\\
[\![p(T,U)]\!]_\sigma &= [\![U]\!]_\sigma \quad \text{if } \sigma(p) = \mathsf{false}.
\end{aligned}$$

The function mapping σ to $[\![T]\!]_\sigma$ is the boolean function described by T. Conversely, i 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

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

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

$$[\![T]\!]_\sigma = [\![U]\!]_\sigma \quad \text{for all } \sigma : A \to \{\mathsf{true}, \mathsf{false}\}.$$

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

(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) $p(x, p(y, z)) = p(x, z)$

for all $p, q \in A$, $p \neq q$. Note that $\mathscr{E}$ is finite if and only if A is finite. Let $\equiv_{\mathscr{E}}$ be the congruence generated by $\mathscr{E}$. We prove that $\mathscr{E}$ is a sound and complete axiomatization for decision equivalence, i.e., the relations $\equiv_{\mathscr{E}}$ 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 $\mathscr{E}$. A straightforward elementary proof is given in [17]; here we give an alternative approach based on rewriting which will be the basis of uniqueness of the ROBDD representation. For the basics of rewriting (confluence, critical pair analysis, termination, recursive path ordering) we refer to [2].

The first step is to complete $\mathscr{E}$: find a confluent and terminating rewrite system DT such that $\equiv_{\mathscr{E}}$ 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 Eq. (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 rules (1) and (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

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

if $p > q$, and

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

if $q > p$, in both cases only using rules (3), (4) and these new rules. We define the rewrite system DT to consist of the rules

$p(x, x)$	$\to$	x	for all p
$p(p(x, y), z)$	$\to$	$p(x, z)$	for all p
$p(x, p(y, z))$	$\to$	$p(x, z)$	for all p
$p(q(x, y), z)$	$\to$	$q(p(x, z), p(y, z))$	for $p > q$
$p(x, q(y, z))$	$\to$	$q(p(x, y), p(x, z))$	for $p > q$.

We have constructed 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. For instance, if $p > q > r$, then the two rules

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

and

$$q(x, r(y, z)) \rightarrow r(q(x, y), q(x, z))$$

give rise to the critical pair

$$\langle p(r(q(x, y), q(x, z)), w),\ q(p(x, w), p(r(y, z), w))\rangle$$

which is convergent due to the reductions

$$\begin{aligned} p(r(q(x, y), q(x, z)), w) &\rightarrow r(p(q(x, y), w), p(q(x, z), w)) \\ &\rightarrow r(q(p(x, w), p(y, w)), p(q(x, z), w)) \\ &\rightarrow r(q(p(x, w), p(y, w)), q(p(x, w), p(z, w))) \end{aligned}$$

and

$$\begin{aligned} q(p(x, w), p(r(y, z), w)) &\rightarrow q(p(x, w), r(p(y, w), p(z, w))) \\ &\rightarrow r(q(p(x, w), p(y, w)), q(p(x, w), p(z, w))). \end{aligned}$$

The full critical pair analysis can be done either by hand or automatically, for the latt approach it has to be remarked that it suffices to prove it for the case of $\#A = 3$ sii no rule contains more than two different symbols. Since all critical pairs converge DT 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 i 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 $U \simeq$ true we obtain $U_1 \simeq$ true and $U_2 \simeq$ true; from the induction hypothesis we conclude $U_1 =$ 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 occur in any of the trees T_1, T_2 and U. For arbitrary σ satisfying $\sigma(p) = \text{true}$ we obtain $[\![T_1]\!]_\sigma = [\![p(T_1, T_2)]\!]_\sigma = [\![U]\!]_\sigma$. Since p does not occur on T_1 and U, the values of $[\![T_1]\!]_\sigma$ and $[\![U]\!]_\sigma$ do not depend on $\sigma(p)$. Hence $[\![T_1]\!]_\sigma = [\![U]\!]_\sigma$ for all σ, hence $T_1 \simeq U$. By taking σ satisfying $\sigma(p) = \text{false}$ we obtain $T_2 \simeq U$ in the same way. 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. For arbitrary s satisfying $\sigma(p) = \text{true}$ we obtain $[\![T_1]\!]_\sigma = [\![p(T_1, T_2)]\!]_\sigma = [\![p(U_1, U_2)]\!]_\sigma = [\![U_1]\!]_\sigma$. Since p does not occur on T_1 and U_1, the values of $[\![T_1]\!]_\sigma$ and $[\![U]\!]_\sigma$ do not depend on $\sigma(p)$. Hence $[\![T_1]\!]_\sigma = [\![U_1]\!]_\sigma$ for all σ, hence $T_1 \simeq U_1$. By taking σ satisfying $\sigma(p) = \text{false}$ we similarly obtain $T_2 \simeq U_2$. 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. For the 'if'-part (completeness) assume $T \simeq U$. Let T', U' be the canonical form of T, U, respectively. By soundness we conclude $T \simeq T'$ and $U \simeq U'$; transitivity of $\simeq$ yields $T' \simeq U'$. By Lemma 3 we conclude $T' = U'$; from Theorem 2 we conclude $T \equiv_\mathcal{E} U$. □

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, in Example 5 we shall see that 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 [17].

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

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

for $i = 1, \ldots, n$, and $V = r(T_n, U_n)$. Clearly V is a decision tree of size $\#V = 4n + 1$. In [18] 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 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. Allowing sharing in the representation admits efficient representation of logical circuits.

Sharing can be done in many ways, but every term has a unique representation as a DAG with maximal sharing, meaning that sharing is applied whenever possible. Every node in this maximally shared DAG represents a subterm of the original term, and subterms are equal if and only if they are represented by the same node. For a term t write $\#_{sh}(t)$ for the number of nodes of the maximally shared DAG representation of t, which we call the shared size of t. Then by the above observation we have

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

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

Example 6 (*See Fig.* 1). Define $T_0 = \mathsf{true}$ and $U_0 = \mathsf{false}$. For binary symbols p_1, p_2, p_3, ... 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 [14]. In [10] 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 destructing 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

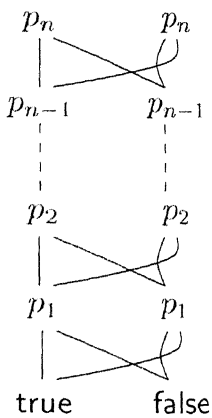

Fig. 1. The effect of sharing: T_n and U_n.

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 [10] 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 7. 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 8. *If $t \Rightarrow t'$, then $t \to^+ t'$.*

Lemma 9. *If $t \to t'$, then a term t'' exists satisfying $t' \to^* t''$ and $t \Rightarrow t''$.*

The following 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 10.
(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 $\to$ is confluent and terminating, then $\Rightarrow$ is confluent and terminating too.*

Proof. Part (1) follows directly from Lemma 8.

If t is a normal form with respect to $\rightarrow$, then it is a normal form with respect to $\Rightarrow$ by Lemma 8. If t is a normal form with respect to $\Rightarrow$, then it is a normal form with respect to $\rightarrow$ by Lemma 9. Hence we have proved part (2).

For part (3) we obtain termination by part (1); it suffices to prove confluence. Assume $s \Rightarrow^* s_1$ and $s \Rightarrow^* s_2$. Since $\Rightarrow$ is terminating 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 $\rightarrow$; by Lemma 8 we have $s \rightarrow^* n_i$ for $i = 1, 2$. From confluence of $\rightarrow$ conclude $n_1 = n_2$. Since $s_i \Rightarrow^* n_i$ for $i = 1, 2$ we proved that $\Rightarrow$ is confluent. □

Note that Theorem 10 holds for any two abstract reduction systems $\rightarrow$ and $\Rightarrow$ satisfying Lemmas 8 and 9 since the proof does not use anything else.

Example 11 (*Due to Vincent van Oostrom*). Not for all assertions in Theorem 10 the converse holds. For instance, the rewrite system consisting of the two rules $f(0, 1) \rightarrow f(1, 1)$ and $1 \rightarrow 0$ admits an infinite reduction $f(0, 1) \rightarrow f(1, 1) \rightarrow f(0, 1) \rightarrow \cdots$, but the shared rewrite relation $\Rightarrow$ is terminating.

For preservation of confluence the combination of termination is essential, as is shown by the rewrite system consisting of the two rules $0 \rightarrow f(0, 1)$ and $1 \rightarrow f(0, 1)$. 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 $l \rightarrow r$ 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. More general, a (non-deterministic) reduction strategy is defined to be 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 $f(x) \rightarrow x$, $a \rightarrow b$ 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 is defined to be a decision tree in which sharing is allowed. An ordered BDD (OBDD) 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 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 12. 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 labeled 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 sharing.

Theorem 13. *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. □

4.1. ROBDDs by rewriting

Next we describe how an arbitrary propositional formula or circuit 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 $\rightarrow$.

For a rewriting approach we already have the TRS DT to transform an arbitrary decision tree to its canonical form. For handling propositional formulas we still need rules to eliminate the propositional connectives. It turns out that by taking care of the order on A in these rules we can achieve that the ultimate decision tree is in canonical form without referring to DT. Moreover, in our approach the standard BDD algorithms based on Bryant's *apply*-function can be mimicked. This *apply*-function computes the ROBDD of $T \Diamond U$ for ROBDDs T and U and binary propositional operations $\Diamond$ in complexity $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. One of our goals is to develop a reduction strategy by which the number of rewrite steps required is of the same order as the complexity of Bryant's basic BDD algorithm. Although the execution of a rewriting step and maintaining maximal sharing may take more than constant time, counting the number of rewriting steps is the most natural notion of complexity of a rewriting computation.

We assume that the propositional formula is constructed from boolean atoms from a finite set A, the values true and false, the unary operation $\neg$ and binary operations $\vee$, $\wedge$ and $\leftrightarrow$, all with their usual meaning. Other operations like implication and exclusive or 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 $\leftrightarrow$: generally formulas including $\leftrightarrow$ or xor cannot be represented in formulas of the same (unshared) complexity without them.

Usually in propositional formulas the boolean atoms appear as constants. In order to fit in the framework of BDDs it is more natural to consider boolean atoms as binary symbols, where a boolean atom p in a formula has to be interpreted as $p(\mathsf{true}, \mathsf{false})$. The replacement of every occurrence of an atom p in the formula by $p(\mathsf{true}, \mathsf{false})$ can be seen as a kind of preprocessing before the rewriting process starts, which is left implicit in the sequel. In this way both propositional formulas and BDDs are represented as terms over the same signature consisting of constants true and false, the unary symbol $\neg$ and in which all

elements of A and the symbols $\vee$, $\wedge$ and $\leftrightarrow$ are binary symbols. This general kind of terms describing boolean functions both covering propositional formulas and BDDs appeared before in [1]. Now we present a rewrite system $\mathscr{B}$ 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 $\leftrightarrow$ we keep the infix notation as is usual in propositional formulas.

Definition 14. The rewrite system $\mathscr{B}$ consists of the following rules, split up into *idempotence rules*, *propagation rules* and *elimination rules*:

$$p(x,x) \to x \quad \text{for all } p \quad \text{(idempotence rules)}$$

$$\left.\begin{array}{rcll} \neg p(x,y) & \to & p(\neg x, \neg y) & \text{for all } p \\ p(x,y) \Diamond p(z,w) & \to & p(x \Diamond z, y \Diamond w) & \text{for all } \Diamond, p \\ p(x,y) \Diamond q(z,w) & \to & p(x \Diamond q(z,w), y \Diamond q(z,w)) & \text{for all } \Diamond, p < q \\ q(x,y) \Diamond p(z,w) & \to & p(q(x,y) \Diamond z, q(x,y) \Diamond w) & \text{for all } \Diamond, p < q \end{array}\right\} \text{(propagation rules)}$$

$$\left.\begin{array}{rcl@{\qquad}rcl} \neg\mathsf{true} & \to & \mathsf{false} & \mathsf{true} \wedge x & \to & x \\ \neg\mathsf{false} & \to & \mathsf{true} & x \wedge \mathsf{true} & \to & x \\ \mathsf{true} \vee x & \to & \mathsf{true} & \mathsf{false} \wedge x & \to & \mathsf{false} \\ x \vee \mathsf{true} & \to & \mathsf{true} & x \wedge \mathsf{false} & \to & \mathsf{false} \\ \mathsf{false} \vee x & \to & x & \mathsf{true} \leftrightarrow x & \to & x \\ x \vee \mathsf{false} & \to & x & x \leftrightarrow \mathsf{true} & \to & x \\ & & & \mathsf{false} \leftrightarrow x & \to & \neg x \\ & & & x \leftrightarrow \mathsf{false} & \to & \neg x \end{array}\right\} \text{(elimination rules)}$$

Here p and q range over A and $\Diamond$ ranges over the symbols $\vee$, $\wedge$ and $\leftrightarrow$.

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

The rewrite system $\mathscr{B}$ 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 $\leftrightarrow \succ \neg$ and $\Diamond \succ p$ for $\Diamond \in \{\neg, \vee, \wedge, \leftrightarrow\}$ 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 $\neg, \vee, \wedge, \leftrightarrow$.

The rewrite system $\mathscr{B}$ 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 $\mathscr{B}$ admits ground normal forms that are not in canonical form. However, when starting with a propositional formula this cannot happen due to the invariance of the following notion of order:

Definition 15. A term is called ordered if for every subterm of the shape $p(T,U)$ for $p \in A$ all symbols $q \in A$ occurring in T or U satisfy $p < q$.

We have the following invariance lemma.

Lemma 16.

- *Every propositional formula is ordered.*
- *If T is ordered and $T \to_{\mathscr{B}}^{*} U$, then U is ordered too.*

Proof. The first claim is clear since in a propositional formula $T = \mathsf{true}$ and $U = \mathsf{false}$ for every subterm of the shape $p(T, U)$. The second claim follows by induction from the same claim without '$*$', which follows from an analysis of the shape of the rules. □

Theorem 17. *Let Φ be a propositional formula over A. Then any reduction of Φ with respect to $\Rightarrow_{\mathscr{B}}$ leads to the same normal form, and this normal form is the unique ROBDD of Φ.*

Proof. From Lemma 16 it follows that normal forms of propositional formulas are ordered. By definition a term is in canonical form if and only if it is ordered and it is in normal form with respect to the idempotence rules. Hence normal forms of propositional formulas are in canonical form, hence are ROBDDs. Since all rewriting steps preserve the represented boolean function, the normal form represents the same boolean function as the original formula. By Theorem 13 the ROBDD representation is unique. □

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 [4,12,16], 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. In the next subsections we discuss and develop a number of strategies.

4.2. *Leftmost innermost reduction*

The simplest and most used rewriting strategy is the leftmost innermost strategy: reduce the leftmost of all innermost redexes. By elaborating an example we now will show that the leftmost innermost strategy, even when adapted to shared rewriting, may be extremely inefficient.

Example 18. As in Example 6 (Fig. 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 the term T_n is the ROBDD of $\leftrightarrow_{i=1}^{n} p_i$ and U_n is the ROBDD of $\neg(\leftrightarrow_{i=1}^{n} p_i)$. In fact they describe the *parity* functions: T_n (respectively, U_n) holds if and only if the number of i-s for which p_i does not hold is even (respectively, odd).

Surprisingly, the leftmost innermost reductions of $\neg(T_n)$ and $\neg(U_n)$ to normal form have exponential lengths as we see in the next proposition. Define a $\neg$-step to be an application of a rule $\neg p(x, y) \to p(\neg x, \neg y)$.

Proposition 19. *For every n both for $\neg(T_n)$ and $\neg(U_n)$ $\Rightarrow_{\mathscr{B}}$-reduction to normal form by the leftmost innermost strategy requires $2^n - 1$ $\neg$-steps.*

Proof. We apply induction on n. For $n = 0$ the proposition trivially holds. For $n > 0$ the first reduction step is

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

The leftmost-innermost reduction continues by reducing $\neg(T_{n-1})$. During this reduction no $\neg\neg$-redex is shared in $\neg(U_{n-1})$ since $\neg(U_{n-1})$ contains only one $\neg\neg$-symbol that is too high in the tree. Hence $\neg(T_{n-1})$ is reduced to normal form with $2^{n-1}-1$ $\neg\neg$-steps due to the induction hypothesis, without affecting the right part $\neg(U_{n-1})$ of the term. After that another $2^{n-1}-1$ $\neg\neg$-steps are required to reduce $\neg(U_{n-1})$, making the total of 2^n-1 $\neg\neg$-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. This exponential behavior of leftmost innermost reduction is caused by the fact that, despite of sharing, during the reduction very often the same redex is reduced.

Since leftmost innermost reduction of the term representation of $\neg(\leftrightarrow_{i=1}^{n} p_i)$ starts with a reduction to $\neg(T_n)$, we see that indeed the leftmost-innermost $\mathscr{B}$-computation of the ROBDD of a propositional formula can be exponential in the size of the formula.

4.3. Layerwise reduction

In this subsection we introduce a new strategy called *layerwise* avoiding the exponential behavior discussed in the previous subsection in which very often the same redex is reduced. More precisely, we show that in layerwise reduction of a term of the shape $\neg T$ or $T \Diamond U$ where T, U are ROBDDs and $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$, every propagation redex is reduced at most once. Ultimately this will lead to a complexity comparable with the standard algorithm for computing ROBDDs. First we investigate the involved redexes.

A redex is called *essential* if it is a propagation redex or an elimination redex. The smallest symbol in a term t containing at least one symbol $p \in A$ is called the *level* of t. For ordered terms it is clear that redexes with respect to the propagation rules as they occur in Definition 14 are exactly the propagation redexes of level p.

Proposition 20. *Let T, U be ROBDDs.*

- *If $\neg T \rightarrow_{\mathscr{B}}^{*} V$, then every essential redex in V is of the shape $\neg T'$ for some subterm T' of T.*
- *If $T \Diamond U \rightarrow_{\mathscr{B}}^{*} V$ 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 $T \leftrightarrow U \rightarrow_{\mathscr{B}}^{*} V$, then every essential redex in V is of the shape $T' \leftrightarrow U'$ or $\neg T'$ or $\neg U'$ for some subterm T' of T and some subterm U' of U.*

Proof. We apply induction on the reduction length of $\rightarrow_{\mathscr{B}}^{*}$. If this length is zero all assertions are trivial. For the induction step we assume that in

$$\neg T \rightarrow_{\mathscr{B}}^{*} V' \rightarrow_{\mathscr{B}} V \quad \text{or} \quad T \Diamond U \rightarrow_{\mathscr{B}}^{*} V' \rightarrow_{\mathscr{B}} V$$

every essential redex in V' is of the desired shape and we have to prove that the same holds for V. If $V' \rightarrow_{\mathscr{B}} V$ is an application of a propagation rule or elimination rule this follows from the shape of the rules; if $V' \rightarrow_{\mathscr{B}} V$ is an idempotence step this follows from the observation that idempotence will not create new essential redexes, note that by definition idempotence cannot be applied on subterms of ROBDDs. □

A term t is called *flat* if $u \to_{\mathscr{B}}^* t$ for $u = \neg T$ for some ROBDD T or $u = T \Diamond U$ for ROBDDs T, U and $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$. The motivation for calling this flat is that in flat terms no connective symbols $\neg, \vee, \wedge$ and $\leftrightarrow$ may occur nested by Proposition 20.

Now we give some lemmas investigating how redexes can be created, and which will motivate the definition of layerwise reduction.

Lemma 21. *Let t be a flat term and let $t \to_{\mathscr{B}} u$ be a reduction for which u contains a propagation redex of level q that is not contained in t, for some $q \in A$. Then either*

- *$t \to_{\mathscr{B}} u$ is a propagation step of level p for $p < q$, or*
- *$t = C[\mathsf{false} \leftrightarrow q(u_1, u_2)]$ or $t = C[q(u_1, u_2) \leftrightarrow \mathsf{false}]$, and $u = \neg(q(u_1, u_2))$ for some context C and some terms u_1, u_2.*

Proof. Let $u = C[l^\sigma]$, where l^σ is the propagation redex of level q not contained in t_2. Let $\Diamond \in \{\neg, \vee, \wedge, \leftrightarrow\}$ be the root of l^σ. By Proposition 20 and the observation that every closed term having its root in $\{\neg, \vee, \wedge, \leftrightarrow\}$ is an essential redex, we see that the position of the redex in t is not below the position of l^σ in u. We distinguish the three possibilities for $t \to_{\mathscr{B}} u$.

- $t \to_{\mathscr{B}} u$ is an idempotence step $C'[p(v, v)] \to C'[v]$. As observed v is not inside l^σ, by which l^σ already occurs in t, contradicting the assumption.
- $t \to_{\mathscr{B}} u$ is a propagation step. We see from the shape of the rules that the only way that l^σ can be created is by a propagation step of level p for $p < q$.
- $t \to_{\mathscr{B}} u$ is an elimination step. We see from the shape of the rules that the only way that l^σ can be created is if $t = C[\mathsf{false} \leftrightarrow q(u_1, u_2)]$ or $t = C[q(u_1, u_2) \leftrightarrow \mathsf{false}]$, and $u = \neg(q(u_1, u_2))$ for some context C and some terms u_1, u_2. □

Lemma 22. *Let t be a flat term and let $t \to_{\mathscr{B}} u$ be a reduction for which u contains a redex of the shape $\mathsf{false} \leftrightarrow q(v, w)$ or $q(v, w) \leftrightarrow \mathsf{false}$ that is not contained in t. Then $t \to_{\mathscr{B}} u$ is a propagation step of level p for $p < q$.*

Proof. By Proposition 20 and the shape of the rules. □

Definition 23. A redex is called *layerwise* if it is not a propagation redex of level q such that

- there is a propagation redex of level p for $p < q$, or
- there is a redex of the shape $\mathsf{false} \leftrightarrow p(t, u)$ or $p(t, u) \leftrightarrow \mathsf{false}$ for $p \leqslant q$.

A layerwise reduction is a reduction reducing only layerwise redexes.

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. By definition a redex is layerwise if and only if every occurrence of this redex is layerwise. Note that a similar property holds for innermost redexes, but not for outermost redexes.

Note that in Definition 23 the second condition only plays a role for reduction of terms in which the symbol $\leftrightarrow$ occurs.

In an earlier version of this paper [15] we proposed a more restricted definition of layerwise in which reducing elimination redexes was often forced to be postponed. Experiments

however show that first reducing elimination redexes often leads to shorter reductions, and sometimes to much shorter reductions.

We will show that layerwise reduction leads to normal forms efficiently for suitable terms.

For a term t its *active level* $\mathsf{actlev}(t)$ is defined to be the smallest value p with respect to $<$ for which t contains either a propagation redex of level p or a redex of the shape $\mathsf{false} \leftrightarrow p(t,u)$ or $p(t,u) \leftrightarrow \mathsf{false}$. If a term does not contain such a redex then its active level is defined to be ∞, with $\infty > p$ for all $p \in A$.

Lemma 24. *Let t be a flat term and let $t \to_{\mathcal{B}} u$. Then* $\mathsf{actlev}(t) \leqslant \mathsf{actlev}(u)$.

Proof. Follows from Lemmas 21 and 22. □

Proposition 25. *In every layerwise $\Rightarrow_{\mathcal{B}}$-reduction of a flat term every propagation redex is reduced at most once.*

Proof. Assume that a propagation redex l^σ of level p is reduced twice:

$$C[l^\sigma] \Rightarrow_{\mathcal{B}} t \Rightarrow_{\mathcal{B}}^* C'[l^\sigma] \Rightarrow_{\mathcal{B}} \cdots$$

Since the reduction is layerwise a propagation redex of level p may only be reduced if the active level is exactly p, hence $\mathsf{actlev}(C[l^\sigma]) = \mathsf{actlev}(C'[l^\sigma]) = p$. By Lemma 24 we conclude that all intermediate terms have active level p too. Since in $C[l^\sigma] \Rightarrow t$ all occurrences of l^σ are reduced, the redex l^σ does not occur in t. Hence the (unshared) reduction $t \to_{\mathcal{B}}^* C'[l^\sigma]$ contains a step $t_1 \to_{\mathcal{B}} t_2$ such that the redex l^σ does not occur in t_1 but occurs in t_2. Lemma 21 allows two cases. The first is that $t_1 \to_{\mathcal{B}} t_2$ is a propagation step of level less than p, contradicting $\mathsf{actlev}(t_1) = p$. The remaining case is that t_1 contains a redex of the shape $\mathsf{false} \leftrightarrow p(u,v)$ or $p(u,v) \leftrightarrow \mathsf{false}$. In none of the reduction steps of $C[l^\sigma] \to_{\mathcal{B}}^* t_1$ this redex can be created according to Lemma 22 since all terms have active level p. Hence the redex $\mathsf{false} \leftrightarrow p(u,v)$ or $p(u,v) \leftrightarrow \mathsf{false}$ already occurs in $C[l^\sigma]$, contradicting the assumption that the first reduction step is layerwise. □

Proposition 25 implies a bound on the number of propagation steps in a layerwise reduction. The following lemma will be used to achieve a bound on the number of all rewrite steps. Since there are rules by which a $\leftrightarrow$-symbol is replaced by a $\neg$-symbol we will only achieve a decrease of size if a $\leftrightarrow$-symbol contributes more to the size than a $\neg$-symbol. That's why for any term t we define $\mathsf{size}(t)$ to be $\#_{sh}(t)$ plus the number of distinct subterms of t having $\leftrightarrow$ as its root, i.e., $\mathsf{size}(t)$ is the number of distinct subterms of t where subterms having $\leftrightarrow$ as its root are counted twice.

Lemma 26. *Let any reduction $t \Rightarrow_{\mathcal{B}}^* u$ consist of m shared propagation steps and n shared idempotence and elimination steps. Then*

$$\mathsf{size}(u) + n \leqslant \mathsf{size}(t) + 3m.$$

Proof. We apply induction on the reduction length $m + n$. If $m + n = 0$, then the claim is trivial. For the induction step assume that the reduction is of the shape $t \Rightarrow_{\mathcal{B}}^* u' \Rightarrow_{\mathcal{B}} u$. By the induction hypothesis we may assume that the claim holds for $t \Rightarrow_{\mathcal{B}}^* u'$; we distinguish two cases:

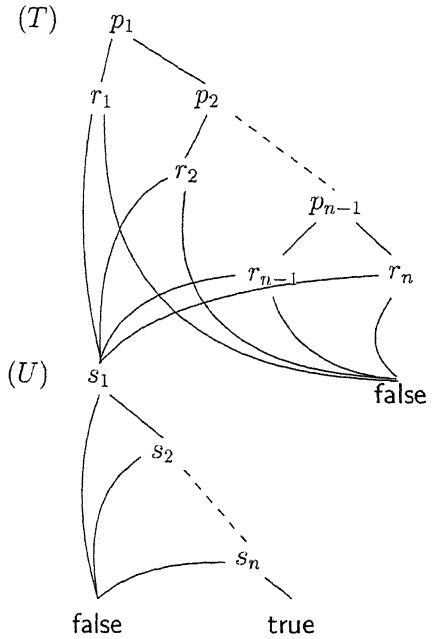

Fig. 2. Counter example for generalized layerwise reduction.

$$
\begin{aligned}
T = {} & p_1(r_1(U, \mathsf{false}),\\
& p_2(r_2(U, \mathsf{false}),\\
& \vdots\\
& p_{n-1}(r_{n-1}(U, \mathsf{false}), r_n(U, \mathsf{false})\cdots).
\end{aligned}
$$

Clearly T is an ROBDD and $\#_{sh}(T) = \mathcal{O}(n)$. A layerwise reduction of $T \leftrightarrow q(\mathsf{true}, \mathsf{false})$ will arrive at $T' = p_1(S_1, p_2(S_2, \ldots, p_{n-1}(S_{n-1}, S_n)\cdots))$ where $S_i = q(r_i(U, \mathsf{false}), r_i(U, \mathsf{false}) \leftrightarrow \mathsf{false})$ for all i. Note that T' does not contain any propagation redex. If the reduction continues by successively reducing every subterm S_i to normal form, then every reduction step satisfies the first condition of Definition 23, but not the second. Since every reduction of S_i to normal form requires $\Omega(n)$ steps, the length of this total reduction of $T \leftrightarrow q(\mathsf{true}, \mathsf{false})$ is $\Omega(n^2)$. For a layerwise reduction this length is $\mathcal{O}(n)$ by Theorem 27. We conclude that the second condition of Definition 23 is essential for Theorem 27 to hold.

Bryant's *apply*-function to compute the ROBDD of $T \Diamond U$ for ROBBDs T and U and a binary operator $\Diamond$ has complexity $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, see [4,12,16]. This is exactly the same bound as the bound on the number of rewrite steps we derived in Theorem 27 for computing the same ROBDD by means of layerwise reduction. Write $\mathsf{apply}(T)$ for layerwise reducing a term T to normal form. We now can define an algorithm reduce to find the ROBDD for a propositional formula Φ:

$$
\begin{aligned}
&\mathsf{reduce}(\mathsf{true}) = \mathsf{true}\\
&\mathsf{reduce}(\mathsf{false}) = \mathsf{false}
\end{aligned}
$$

- Let $u' \Rightarrow_{\mathscr{B}} u$ be a propagation step. Due to the shape of the rules we conclude that $\mathsf{size}(u) \leqslant \mathsf{size}(u') + 3$. Here equality may hold for propagation rules for $\leftrightarrow$. Combined with the induction hypothesis $\mathsf{size}(u') + n \leqslant \mathsf{size}(t) + 3(m-1)$ we obtain $\mathsf{size}(u) + n \leqslant \mathsf{size}(t) + 3m$.
- Let $u' \Rightarrow_{\mathscr{B}} u$ be an idempotence or elimination step. Due to the shape of the rules we conclude that $\mathsf{size}(u) \leqslant \mathsf{size}(u') - 1$ (here is the point where we need size instead of $\#_{sh}$: by the rules $\mathsf{false} \leftrightarrow x \to \neg x$ and $x \leftrightarrow \mathsf{false} \to \neg x$ the shared size $\#_{sh}$ does not strictly decrease). Combined with the induction hypothesis $\mathsf{size}(u') + (n-1) \leqslant \mathsf{size}(t) + 3m$ we obtain $\mathsf{size}(u) + n \leqslant \mathsf{size}(t) + 3m$.

In both cases we are done. □

Theorem 27. *Let T be a ROBDD. Then every layerwise $\Rightarrow_{\mathscr{B}}$-reduction of $\neg T$ contains $\mathcal{O}(\#_{sh}(T))$ steps.*

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

Proof. Let the considered reduction consist of m shared propagation steps and n shared idempotence and elimination steps. By Proposition 20, the number of candidates for propagation redexes is $\mathcal{O}(\#_{sh}(T))$ or $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, respectively. By Proposition 25 each of these candidates is reduced at most once. Hence the total number m of propagation steps has the required complexity. Applying Lemma 26 we conclude that the total number of steps $m + n$ is less than or equal to $4m + \mathsf{size}(t)$. If $t = \neg T$, we have $\mathsf{size}(t) = \mathcal{O}(\#_{sh}(T))$ and we already observed that $m = \mathcal{O}(\#_{sh}(T))$, hence $m + n = \mathcal{O}(\#_{sh}(T))$. If $t = T \Diamond U$, we have $\mathsf{size}(t) = \mathcal{O}(\#_{sh}(T) + \#_{sh}(U))$ and we already observed that $m = \mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, hence $m + n = \mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. For both cases this concludes the proof. □

In Example 18, we saw that the first condition of Definition 23 is essential for Proposition 25 and Theorem 27 to hold. In the following two examples we show that the same holds for the second condition of Definition 23.

Example 28. Let $p < q$ and consider the following reduction in which the same q-redex $\neg(q(\mathsf{true}, \mathsf{false}))$ is reduced twice:

$$
\begin{aligned}
& p\big(\mathsf{false}, q(\mathsf{true}, \mathsf{false})\big) \leftrightarrow p\big(q(\mathsf{true}, \mathsf{false}), \mathsf{false}\big) \\
& \quad \Rightarrow_{\mathscr{B}} p\big(\mathsf{false} \leftrightarrow q(\mathsf{true}, \mathsf{false}), q(\mathsf{true}, \mathsf{false}) \leftrightarrow \mathsf{false}\big) \\
& \quad \Rightarrow_{\mathscr{B}} p\big(\neg(q(\mathsf{true}, \mathsf{false})), q(\mathsf{true}, \mathsf{false}) \leftrightarrow \mathsf{false}\big) \\
& \quad \Rightarrow_{\mathscr{B}} p\big(q(\neg(\mathsf{true}), \neg(\mathsf{false})), q(\mathsf{true}, \mathsf{false}) \leftrightarrow \mathsf{false}\big) \\
& \quad \Rightarrow_{\mathscr{B}} p\big(q(\neg(\mathsf{true}), \neg(\mathsf{false})), \neg(q(\mathsf{true}, \mathsf{false}))\big) \\
& \quad \Rightarrow_{\mathscr{B}} p\big(q(\neg(\mathsf{true}), \neg(\mathsf{false})), q(\neg(\mathsf{true}), \neg(\mathsf{false}))\big).
\end{aligned}
$$

Every reduction step satisfies the first condition of Definition 23, but not the second. Hence the second condition of Definition 23 is essential for Proposition 25 to hold.

Example 29 (*See Fig.* 2). Let $p_1 < \cdots < p_{n-1} < q < r_1 < \cdots < r_n < s_1 < \cdots < s_n$, let $U = s_1(\mathsf{false}, s_2(\mathsf{false}, \ldots, s_n(\mathsf{false}, \mathsf{true}) \cdots))$, and let

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

Roughly speaking this function reduce mimics the standard algorithm from [4,12,16]. Note that applying reduce is not pure rewriting but a combination of rewriting and function application. In the following section we succeed in mimicking the standard algorithm by pure rewriting.

4.4. *Layerwise innermost reduction*

In order to compute the ROBDD of a propositional formula Φ we simply want to rewrite Φ by $\mathscr{B}$ using a particular reduction strategy, until the normal form is obtained which is the desired ROBDD according to Theorem 17. In this subsection we introduce a *layerwise innermost* strategy for which we show that the required number of rewrite steps does not exceed the known complexity bound of the standard algorithm.

Definition 30. A redex is called *layerwise innermost* if it is an innermost redex that is not a propagation redex of level q such that
- there is an innermost redex with respect to a propagation rule of level p for $p < q$, or
- there is an innermost redex of the shape $\mathsf{false} \leftrightarrow p(t,u)$ or $p(t,u) \leftrightarrow \mathsf{false}$ for $p \leqslant q$.

A layerwise innermost reduction is a reduction reducing only layerwise innermost redexes.

Clearly every term not in normal form contains a layerwise innermost redex, and just like innermost, outermost and layerwise, layerwise innermost reduction is a non-deterministic reduction strategy. The following lemma gives the properties of layerwise innermost reduction that we need in the sequel.

Lemma 31.
(1) *Every layerwise innermost redex is innermost, i.e., every proper subterm is in normal form.*
(2) *Let T, U be ROBDDs and $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$. Then every layerwise innermost reduction of $\neg T$ or $T \Diamond U$ to normal form is a layerwise reduction too.*
(3) *Let a subterm of a term t be a layerwise innermost redex of $\neg t$ or $t \Diamond u$ or $u \Diamond t$. Then it is a layerwise innermost redex of t too.*

Proof. Parts (1) and (3) are immediate from the definition; part (2) holds since every essential redex occurring in a reduction of $\neg T$ or $T \Diamond U$ to normal form is an innermost redex according to Proposition 20. □

In order to compare the reduction lengths of layerwise innermost reduction with the complexity bound of the standard algorithm, we first specify that complexity bound. Fix an order $<$ on A. Let F be recursively defined on propositional formulas as follows.

$$\begin{aligned} F(p) &= 0 && \text{for } p \in A \\ F(\neg\phi) &= F(\phi) + s(\phi) \\ F(\phi \Diamond \psi) &= F(\phi) + F(\psi) + s(\phi)s(\psi) && \text{for } \Diamond \in \{\vee, \wedge, \leftrightarrow\}, \end{aligned}$$

where $s(\phi)$ is defined to be the shared size of the ROBDD of ϕ with respect to $<$. The standard *apply*-function to compute the ROBDD of $T \Diamond U$ for ROBBDs T and U and a binary operator has complexity $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. As consequence, there is a constant C such that the number of computation steps to compute the ROBDD of a propositional formula ϕ by means of the standard algorithm is at most $C * F(\phi)$. Our main theorem states that the number of reduction steps of a layerwise innermost reduction of ϕ has exactly the same complexity bound.

Before presenting and proving the main theorem we give a lemma.

Lemma 32. *Let $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$ and let $t \Diamond u \Rightarrow_{\mathscr{B}}^* t' \Diamond u'$ be a layerwise innermost reduction of n steps not containing any root reduction step. Then there are layerwise innermost reductions $t \Rightarrow_{\mathscr{B}}^* t'$ and $u \Rightarrow_{\mathscr{B}}^* u'$ of n_t, n_u steps, respectively, satisfying $n \leqslant n_t + n_u$.*

Proof. We apply induction on n. For $n = 0$ the lemma trivially holds. For $n > 0$ the reduction can be written as

$$t \Diamond u \Rightarrow_{\mathscr{B}} t'' \Diamond u'' \Rightarrow_{\mathscr{B}}^* t' \Diamond u'.$$

By the induction hypothesis we have layerwise innermost reductions $t'' \Rightarrow_{\mathscr{B}}^* t'$ and $u'' \Rightarrow_{\mathscr{B}}^* u'$ of n_1, n_2 steps, respectively, satisfying $n - 1 \leqslant n_1 + n_2$. For the redex of the first step $t \Diamond u \Rightarrow_{\mathscr{B}} t'' \Diamond u''$ we have three possibilities:

- It occurs in t and not in u. Then $u = u''$ and there is a layerwise innermost reduction step $t \Rightarrow_{\mathscr{B}} t''$, giving rise to layerwise innermost reductions $t \Rightarrow_{\mathscr{B}}^* t'$ and $u \Rightarrow_{\mathscr{B}}^* u'$ of $n_t = n_1 + 1$ and $n_u = n_2$ steps, respectively, hence satisfying $n = 1 + (n - 1) \leqslant 1 + n_1 + n_2 = n_t + n_u$.
- It occurs in u and not in t. Then $t = t''$ and there is a layerwise innermost reduction step $u \Rightarrow_{\mathscr{B}} u''$, giving rise to layerwise innermost reductions $t \Rightarrow_{\mathscr{B}}^* t'$ and $u \Rightarrow_{\mathscr{B}}^* u'$ of $n_t = n_1$ and $n_u = n_2 + 1$ steps, respectively, hence satisfying $n = 1 + (n - 1) \leqslant 1 + n_1 + n_2 = n_t + n_u$.
- It occurs both in t and in u. Then there are layerwise innermost reduction steps $t \Rightarrow_{\mathscr{B}} t''$ and $u \Rightarrow_{\mathscr{B}} u''$, giving rise to layerwise innermost reductions $t \Rightarrow_{\mathscr{B}}^* t'$ and $u \Rightarrow_{\mathscr{B}}^* u'$ of $n_t = n_1 + 1$ and $n_u = n_2 + 1$ steps, respectively, hence satisfying $n = 1 + (n - 1) \leqslant 1 + n_1 + n_2 < n_t + n_u$.

Here the reduction steps $t \Rightarrow_{\mathscr{B}} t''$ and $u \Rightarrow_{\mathscr{B}} u''$ are layerwise innermost by Lemma 31, part (3). In all cases we are done. □

Theorem 33. *There is a constant C such that the computation of the ROBDD of any propositional formula ϕ by applying layerwise innermost reduction to normal form requires at most $C * F(\phi)$ reduction steps.*

Proof. Let C be the constant implied by Theorem 27 such that for every ROBDD T every layerwise $\Rightarrow_{\mathscr{B}}$-reduction of $\neg T$ contains at most $C * \#_{sh}(T)$ steps, and that for all ROBDDS T, U every layerwise $\Rightarrow_{\mathscr{B}}$-reduction of $T \vee U$, $T \wedge U$ or $T \leftrightarrow U$ contains at most $C * \#_{sh}(T) * \#_{sh}(U)$ steps.

We will prove the theorem by induction on the structure of the formula ϕ.

For the basis of the induction we have $\phi = p(\mathsf{true}, \mathsf{false})$ is in normal form, and the claim holds.

For the induction step we distinguish two cases: $\phi = \neg\phi_1$ and $\phi = \phi_1 \Diamond \phi_2$ for $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$.

Let $\phi = \neg\phi_1$. Let T be the ROBDD of ϕ_1 and let U be the ROBDD of ϕ. By Lemma 31, part (1), every layerwise innermost reduction of ϕ is of the shape

$$\phi = \neg\phi_1 \Rightarrow_{\mathscr{B}}^* \neg T \Rightarrow_{\mathscr{B}}^* U.$$

By removing the top $\neg$ symbol in the reduction $\neg\phi_1 \Rightarrow_{\mathscr{B}}^* \neg T$ we obtain a reduction from ϕ_1 to T, which is layerwise innermost by Lemma 31, part (3). Hence it has at most $C * F(\phi_1)$ steps by induction hypothesis. Hence the reduction $\neg\phi_1 \Rightarrow_{\mathscr{B}}^* \neg T$ has at most $C * F(\phi_1)$ steps. The layerwise innermost reduction $\neg T \Rightarrow_{\mathscr{B}}^* U$ is layerwise too by Lemma 31, part (2). Then by the definition of C we conclude that the reduction $\neg T \Rightarrow_{\mathscr{B}}^* U$ consists of at most $C * \#_{sh}(T) = C * s(\phi_1)$ steps. We conclude that the total number of steps in the layerwise innermost reduction of ϕ is at most $C * F(\phi_1) + C * s(\phi_1) = C * F(\neg\phi_1) = C * F(\phi)$.

For the remaining case let $\phi = \phi_1 \Diamond \phi_2$. Let T_i be the ROBDD of ϕ_i for $i = 1, 2$, and let U be the ROBDD of ϕ. By Lemma 31, part (1), every layerwise innermost reduction of ϕ is of the shape

$$\phi = \phi_1 \Diamond \phi_2 \Rightarrow_{\mathscr{B}}^* T_1 \Diamond T_2 \Rightarrow_{\mathscr{B}}^* U.$$

Due to Lemma 32 the number of steps of the first part $\phi_1 \Diamond \phi_2 \Rightarrow_{\mathscr{B}}^* T_1 \Diamond T_2$ is at most $n_1 + n_2$ where n_i is the length of a layerwise innermost reduction $\phi_i \Rightarrow_{\mathscr{B}}^* T_i$ for $i = 1, 2$. By the induction hypothesis we obtain $n_i \leqslant C * F(\phi_i)$ for $i = 1, 2$. The second part $T_1 \Diamond T_2 \Rightarrow_{\mathscr{B}}^* U$ of the reduction is layerwise by Lemma 31, part (2); by the definition of C we conclude that it contains at most $C * \#_{sh}(T_1) * \#_{sh}(T_2)$ steps. Combining these results on the first and second part of the reduction $\phi \Rightarrow_{\mathscr{B}}^* U$ we conclude that the total number of steps is at most $C * F(\phi_1) + C * F(\phi_2) + C * \#_{sh}(T_1) * \#_{sh}(T_2) = C * F(\phi)$. □

4.5. Lazy reduction

With the innermost reduction strategy, the deepest connectives are propagated downwards, and finally eliminated. We will now consider the opposite strategy, called *lazy strategy*, which is defined in such a way that a step is only performed if it contributes to lifting the smallest variable to the root. Technically, the definition proceeds by distinguishing *head reduction* ($\rightarrow_H$) and *lazy reduction* ($\rightarrow_L$). Head reduction will be defined as the closure of propagation and elimination rule instances under connectives. Lazy reduction is the closure of head reduction and the idempotence rule under proposition symbols.

Definition 34. The *head reduction steps* are defined inductively by the following clauses:

- For any elimination and propagation rule $l \rightarrow r$ and ground substitution σ, $l^\sigma \rightarrow_H r^\sigma$.
- If $s \rightarrow_H t$, then $\neg s \rightarrow_H \neg t$ and for any ground term r and connective $\Diamond \in \{\vee, \wedge, \leftrightarrow\}$, $s \Diamond r \rightarrow_H t \Diamond r$ and $r \Diamond s \rightarrow_H r \Diamond t$.

The *lazy reduction steps* are defined inductively as follows:

- For any ground term t, $p(t, t) \rightarrow_L t$.
- If $s \rightarrow_H t$, then $s \rightarrow_L t$.
- If $s \rightarrow_L t$, then for any ground term r and atom p, $p(r, s) \rightarrow_L p(r, t)$ and $p(s, r) \rightarrow_L p(t, r)$.

A redex l^σ is called *lazy* in $C[l^\sigma]$, if we have $C[l^\sigma] \to_L C[r^\sigma]$ for the corresponding right-hand side r. We now show that $\to_L$ is a strategy, in the sense that each term that is not yet in normal form contains a lazy redex. This first requires a lemma on head normal forms. For a rewrite relation $\to$ we write $t \not\to$ to denote that t is in normal form, i.e., cannot be rewritten by $\to$.

Lemma 35. *If $t \not\to_H$ for some closed t, then t is of the form* false, true *or* $p(t', t'')$.

Proof. Induction on t. Let $t \not\to_H$. If $t = r \Diamond s$, for some connective $\Diamond$, then $r \not\to_H$ and $s \not\to_H$, otherwise t would do a $\to_H$-step. By induction hypothesis, s is of the form true, false or $p(s', s'')$, and similar for r. Now for each of the nine cases that arise, there exists some elimination or propagation rule in the TRS $\mathscr{B}$ by which $r \Diamond s$ can do a $\to_H$ step; contradiction. So t cannot be of the form $r \Diamond s$. Similar for $t = \neg r$. □

According to the previous lemma, terms of the form false, true and $p(r, s)$ can be called *head normal forms*. This notion was used in an alternative definition of the lazy strategy in [15]. The current definition is more precise, and the separate notion of head reduction admits a nicer formulation of the main property of lazy reductions (Proposition 37).

Proposition 36. *If $t \not\to_L$, then $t \not\to_{\mathscr{B}}$.*

Proof. Induction on t. Let t be in $\to_L$ normal form. If t is one of true, false, then clearly $t \not\to_{\mathscr{B}}$. Furthermore, t cannot be of the form $r \Diamond s$ or $\neg r$, for then t can do a $\to_H$ step by Lemma 35, which is a $\to_L$ step by definition. Finally, consider t of the form $p(r, s)$. The only rule to do a top-level reduction is idempotence, but this would be a $\to_L$-step which is not possible by assumption. Next, r and s must be in $\to_L$ normal form, for otherwise t does a $\to_L$ step. Hence by induction hypothesis, r and s are in $\to_{\mathscr{B}}$ normal form too, so a $\to_{\mathscr{B}}$ step inside a proper subterm of t is also excluded. □

For arbitrary TRS the lazy reduction strategy can be defined by distinguishing constructor symbols and defined symbols. In our case, false, true and the atoms are the constructor symbols, as they appear in normal forms, and the connectives are the defined symbols. The idempotence rule is special, in the sense that the root symbol of the left-hand side is a constructor symbol. The special treatment of idempotence in our definition of lazy rewriting is motivated by Example 38.

Lazy reduction is related, but not identical, to outermost rewriting. First, outermost idempotence redexes can always be postponed in our definition. Also, true $\wedge$ false is a lazy redex in the term true $\wedge$ (true $\wedge$ false), although not outermost; this is on purpose, as the topmost constructor of the right conjunct is not yet visible. Conversely, in $p(\neg$true, true$) \wedge$ (true $\wedge$ false), the subterm $\neg$true is an outermost redex, but not a lazy redex; this is on purpose, as the topmost constructor of the left conjunct $p(\neg$true, true) is already visible.

Lazy reduction can be lifted to shared rewriting in the usual way: first identify some lazy redex, then replace all its occurrences. Note that opposed to the innermost and the layerwise strategies, some of these occurrences might be in non-lazy position. Consider for instance q(true, $\neg$true) $\wedge \neg$true. We have:

$$q(\mathsf{true}, \neg\mathsf{true}) \wedge \neg\mathsf{true} \;\to_L\; q(\mathsf{true}, \neg\mathsf{true}) \wedge \mathsf{false}$$

but

$$q(\mathsf{true}, \neg\mathsf{true}) \wedge \neg\mathsf{true} \;\not\to_L\; q(\mathsf{true}, \mathsf{false}) \wedge \neg\mathsf{true}$$

Nevertheless, using shared rewriting both copies are rewritten:

$$q(\mathsf{true}, \neg\mathsf{true}) \wedge \neg\mathsf{true} \;\Rightarrow_L\; q(\mathsf{true}, \mathsf{false}) \wedge \mathsf{false}.$$

We will now prove the main property of lazy reduction, which states that the smallest variable is lifted to the top in a number of steps linear to the size of the shared term. This strengthens the result in [15] considerably, which was only proved for the unshared size. The result is stated in terms of head reduction.

Proposition 37. *If* $t \Rightarrow_H^n t'$, *then* $n \leqslant 2.\#_{sh}(t)$.

Proof. First define the set H of subterms of t that are accessible by head reduction as follows:

$$\begin{aligned} &H(\mathsf{true}) = H(\mathsf{false}) = H(p(t,u)) = \emptyset \\ &H(\neg t) = \{\neg t\} \cup H(t) \\ &H(t \Diamond u) = \{t \Diamond u\} \cup H(t) \cup H(u) \quad \text{for } \Diamond \in \{\vee, \wedge, \leftrightarrow\} \end{aligned}$$

Let $\mathsf{top}(t)$ denote the number of nodes in the accessible top, counting $\leftrightarrow$ twice, i.e.,

$$\mathsf{top}(t) = \#H(t) + \#\{u \in H(t) \mid \exists v, w.\, u = v \leftrightarrow w\}$$

Because $\Rightarrow_H$ is defined as the closure of elimination and propagation rules under connectives, each $\Rightarrow_H$ step removes all occurrences of some elimination or propagation redex l^σ from $H(t)$. That redex is replaced by the corresponding right-hand side r^σ, which starts with an atom in all cases, except that $\leftrightarrow$ might be replaced by $\neg$ (for this reason $\leftrightarrow$ was counted twice), effectively decreasing $\mathsf{top}(t)$ with at least one. Other subterms in $H(t)$ are either removed by an elimination rule (decreasing $\mathsf{top}(t)$ further), or they are of the form $C[l^\sigma]$ and replaced by $C[r^\sigma]$ (possibly decreasing $\mathsf{top}(t)$ due to sharing). Hence, if $t \Rightarrow_H u$ then $\mathsf{top}(t) > \mathsf{top}(u)$. Clearly, $\mathsf{top}(t) \leqslant 2.\#_{sh}(t)$. So the number of possible rewrite steps $n \leqslant \mathsf{top}(t) \leqslant 2.\#_{sh}(t)$.

Combined with Lemma 35, we have that *head*-reduction reveals the top-most symbol of the BDD in a linear number of steps. Although the proposition is stated in terms of head reduction, we obtain as a corollary that each lazy reduction sequence starting in t reaches a term of the form $p(u, v)$ in at most $2 * \#_{sh}(t)$ steps, but in the next lazy step p may disappear due to a propagation step.

Note that by applying the idempotence rule the accessible top might increase, and the above proof would not be valid. The following example shows that excluding idempotence from head reductions is indeed essential.

Example 38. Write $p \mathbin{\frown} t$ to denote $p(t, t)$. Consider the shared term

$$\underbrace{\neg \cdots \neg}_{n} p_1 \mathbin{\frown} p_2 \mathbin{\frown} \cdots \mathbin{\frown} p_n \mathbin{\frown} \mathsf{true}.$$

As a shared term, this has size linear in n. We have the following reduction:

$$\begin{aligned}
& \underbrace{\neg\cdots\neg}_{n} p_1 \frown p_2 \frown \cdots \frown p_n \frown \mathsf{true} \\
\Rightarrow_H^{n-1} \quad & \neg p_1 \frown \underbrace{\neg\cdots\neg}_{n-1} p_2 \frown \cdots \frown p_n \frown \mathsf{true} \\
\rightarrow_{idemp} \quad & \underbrace{\neg\cdots\neg}_{n} p_2 \frown \cdots \frown p_n \frown \mathsf{true} \\
& \cdots \\
\rightarrow \quad & \underbrace{\neg\cdots\neg}_{n} \mathsf{true} \\
\Rightarrow_H^{n} \quad & \mathsf{false/true}
\end{aligned}$$

The length of this reduction is quadratic in n. Apart from the idempotence steps it is a head reduction, showing that for validity of the linearity bound of Proposition 37 it is essential to exclude idempotence in the definition of head reduction.

We conclude this section by giving a typical example where lazy reduction outperforms any innermost strategy, like the traditional apply-algorithm.

Example 39. Let Φ be a formula of size m, whose ROBDD-representation is exponentially large in m. Without loss of generality we assume that true and false do not occur in Φ and that some variable p is strictly smaller than all variables occurring in formula Φ. Consider the formula $p \wedge (\Phi \wedge \neg p)$, which is clearly unsatisfiable. Note that any innermost strategy, such as the traditional algorithm using *apply*, will as an intermediate step always completely build the ROBDD for Φ, which is known to be exponential in m.

We now show that the lazy strategy has linear time complexity. Replace each propositional 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:

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

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. Note that the reduction of $\neg p$ doesn't interfere with Φ, as Φ doesn't contain p. By Lemma 37 we have $n \leqslant 2m$. Furthermore, depending on the order in which the redexes are contracted, $4 \leqslant k \leqslant 6$, so the length of the reduction is linear in m. Note that reductions inside Φ_1 and Φ_2 are never permitted.

In [1] the *up-one* algorithm is devised, which brings a variable to the top of a BED in linear time. Our work unifies this algorithm with the usual apply algorithm (called *up-all* for BEDs), by showing that both algorithms can be obtained as a special reduction strategy of the same TRS. Lazy reduction coincides with the repeated application of up-one, and layer-

wise innermost reduction coincides with up-all. In [1] it is noticed that *up-one* can be more effective in case the resulting BDD is small, like in the example above, but there the effect is partially attributed to the addition of logical rewrite rules, such as $x \wedge (y \wedge \neg x) \to \mathsf{false}$. Our example shows that pure lazy rewriting can have similar effects. The effect of rewrite rules acting on nodes labeled with connectives can also be obtained by the level exchange operation on nodes with newly introduced variables [9].

5. Experiments and concluding remarks

In order to get a better impression of the performance of the various strategies we made an implementation for them counting the number of rewrite steps to reach the ROBDD for a number of formulas (see Table 1).

We consider five strategies: leftmost innermost, layerwise, layerwise innermost, leftmost lazy and layerwise lazy. Here the layerwise lazy strategy means that among all lazy redexes one is chosen that satisfies the requirements as they occur in the definition of layerwise.

We consider six formulas. The first one is the fourth pigeon hole formula ph4:

$$\left(\bigwedge_{i=1}^{5}\left(\bigvee_{j=1}^{4} p_{ij}\right)\right) \wedge \left(\bigwedge_{j=1,\ldots,4,1\leqslant i<k\leqslant 5} (\neg p_{ij} \vee \neg p_{kj})\right).$$

Since the first big conjunct implies that at least 5 variables are true and the second big conjunct implies that at most 4 variables are true, the formula is equivalent to false.

The second formula is chess4, a formula describing all possibilities to tile a 4×4 chess board by 8 dominoes.

The third formula is a random formula of size 200 on 10 variables. Without going into detail of how to define a random formula we only mention that we tried a number of samples for different notions of randomness all roughly giving the same pattern as the sample given in the table.

The fourth is bi-imp, a bi-implicational formula consisting only of $\leftrightarrow$:

$p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_{15} \leftrightarrow p_1 \leftrightarrow p_2 \leftrightarrow \cdots \leftrightarrow p_{15}$, associated from left to right. It is not difficult to see that this formula is equivalent to true.

The last two are unsatisfiable formulas $\mathrm{uns}_i = p \wedge (\Phi_i \wedge \neg p)$ for $i = 1, 2$, where $\Phi_1 = \bigvee_{i=1}^{10}(p_i \wedge q_i)$ and $\Phi_2 = (\neg q) \wedge \Phi_1$. We take the order $p < p_1 < p_2 < \cdots < p_{10} < q_1 < q_2 < \cdots < q_{10} < q$. The formula Φ_1 with this order is a standard example of a small formula having a big ROBDD: the ROBDD of Φ_1 has 2046 internal nodes.

Table 1

Strategy	Formula					
	Ph4	Chess4	Random	Bi-imp	uns_1	uns_2
Leftmost innermost	32 633	3248	6962	262 147	4119	16 408
Layerwise	2113	1863	2946	1790	58	4123
Layerwise innermost	4225	3142	2849	633	4119	7202
Leftmost lazy	7017	2024	5319	671 750	29	31
Layerwise lazy	1984	1696	3333	1986	29	31

Every number in the table denotes the number of rewrite steps required for the given formula to reach the ROBDD by using the given rewrite strategy.

It is not possible to choose a definite winner among the given strategies. Roughly speaking we can say that layerwise innermost is often a good strategy, but that sometimes layerwise or layerwise lazy is better. However, the formulas uns_1 and uns_2 serve bad for layerwise innermost (with shared sizes of intermediate terms of over 2000, which does not occur in the other examples) and very well for both lazy strategies. For the example uns_1 the layerwise strategy is not bad; in the example uns_2 however the lazy strategies convincingly outperform the layerwise strategy.

As a conclusion we state that we developed a framework for a wide class of algorithms to compute the ROBDD of a propositional formula, all proved to be correct, essentially covering the standard algorithm, but also covering algorithms that perform much better than the standard algorithm in particular cases.

References

[1] H.R. Andersen, H. Hulgaard, Boolean expression diagrams, in: Twelfth Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, IEEE Computer Society, Silver Spring, MD, 1997, pp. 88–98.

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

[3] M.G.J. van den Brand, H.A. de Jong, P. Klint, P.A. Olivier, Efficient annotated terms. Software – Practice and Experience 30 (3) (2000) 259–291.

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

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

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

[7] E. Clarke, E. Emerson, A. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans. Prog. Languages Syst. 8 (2) (1986) 244–263.

[8] J. Groote, J. van de Pol, Equational binary decision diagrams, in: M. Parigot, A. Voronkov (Eds.), Logic for Programming and Reasoning, LPAR2000, Lecture Notes in Artificial Intelligence, vol. 1955, Springer, Berlin, 2000, pp. 161–178.

[9] A. Hett, R. Drechsler, B. Becker, MORE: Alternative implementation of BDD packages by multi-operand synthesis, in: European Design Automation Conference, 1996, pp. 164–169.

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

[11] J.E. Klop, Term rewriting systems, in: D.G.S. Abramski, T. Maibaum (Eds.), Handbook of Logic in Computer Science, vol. 2, Oxford University Press, Oxford, 1992.

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

[13] J. Møller, J. Lichtenberg, H.R. Andersen, H. Hulgaard, Difference decision diagrams, in: Computer Science Logic, Denmark, 1999.

[14] D. Plump, Term graph rewriting, H. Ehrig, G. Engels, H.J. Kreowski, G. Rozenberg (Eds.), in: Handbook of Graph Grammars and Computing by Graph Transformation, Applications, Languages, vol. 2, World Scientific, Singapore, 1999, pp. 3–61.

[15] J. van de Pol, H. Zantema, Binary decision diagrams by shared rewriting, in: M. Nielsen, B. Rovan (Eds.), Mathematical Foundations of Computer Science, MFCS2000, Lecture Notes in Computer Science, vol. 1893, Springer, Berlin, 2000, pp. 609–618.

[16] I. Wegener, Branching Programs and Binary Decision Diagrams, Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia, PA, 2000.

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

[18] H. Zantema, H.L. Bodlaender, Sizes of ordered decision trees, Int. J. on Found. of Comp. Sci. (2001), to appear (preliminary version appeared as report UU-CS-1999-31, Utrecht University).