Clause Form Conversions for Boolean Circuits

Paul Jackson and Daniel Sheridan

School of Informatics
University of Edinburgh
Edinburgh, UK
pbj@inf.ed.ac.uk
d.j.sheridan@sms.ed.ac.uk

Abstract. Boolean circuits are well established as a data structure for building propositional encodings of problems in preparation for satisfiability solving. The standard method for converting Boolean circuits to clause form (naming every vertex) has a number of shortcomings
In this paper we give a projection of several well-known clause form conversions to a simplified Boolean circuit. We introduce a new conversion which we show is equivalent to that of Boy de la Tour in certain circumstances and is hence optimal in the number of clauses that it produces. We extend the algorithm to cover reduced Boolean circuits, a data structure used by the model checker NuSMV.
We present experimental results for this and other conversion procedures on BMC problems demonstrating its superiority, and conclude that the CNF conversion plays a significant role in reducing the overall solving time.

1 Introduction

SAT solvers based on the DPLL procedure typically require their input to be in conjunctive normal form (CNF). Earlier papers dealing with encoding to SAT, particularly much of the planning literature, encode directly from the input representation to clause form. More recent encoding work makes little mention of CNF conversion. Biere et al., proposing BMC [3], give an encoding to propositional logic only. Similarly, although the SNF encoding for BMC [6] discusses the clauses generated, the majority of the presentation in in general propositional logic. The microprocessor verification work of Velev includes a thorough analysis of improving the clause form generated [11], but the work is not immediately applicable to general propositional logic. Nevertheless, Velev is able to claim a speed up by a factor of 32 by altering the clause form conversion.

There is other evidence to motivate the study of clause form conversions for SAT. While focussing on CNF representations of cardinality constraints, Bailleux and Boufkhad [2] give a reformulation of the parity problems which have been standard SAT benchmarks for a number of years. They argue that the problems are made harder than they should be by a poor clause form representation, and demonstrate a dramatic speedup on the par32 problem with modern solvers on the reformulated problem.

In the first-order logic domain, the CNF conversion problem was handled comprehensively by Boy de la Tour [4]. The algorithm given is impractical without the improvements by Nonnengart et al. [9], but the resulting algorithm is fiddly to implement making it hard to be confident of a correct implementation.

In this paper we introduce a simple and easy to understand CNF conversion algorithm for propositional logic and prove that it is optimal with respect to the number of clauses. As its time complexity is linear, it represents a significant improvement over the (quadratic) Boy de la Tour algorithm. Of course, it is well known that problem size does not necessarily correspond to solving time in SAT, so we present some experimental results demonstrating the effect that our algorithm has on some BMC [3] problems.

1.1 Notation conventions

In an attempt to improve the clarity of the presentation, we use a number of conventions in our notation. Much of the work is concerned with both graphs and propositional logic, so we distinguish between *graph variables* ranging over vertices and edges given in italic capitals (X, Y) and *propositional variables* given in italic lower case (x, y); vertices are typically denoted V and edges E and this notation is significant in determining the type of a function. We will use the shorthand of referring to a subgraph by a single edge; the subgraph thus identified includes all of the descendents of the edge given, and such an edge is called the *root* of the subgraph and denoted T. Sets of vertices or edges are given in bold type ($\mathbf{X}$, $\mathbf{Y}$).

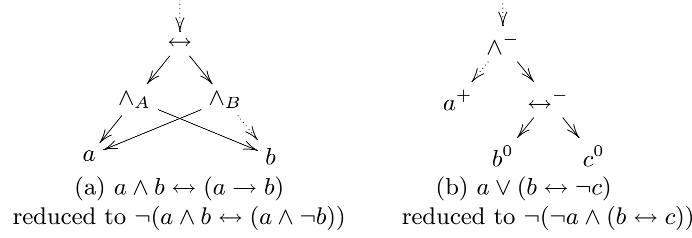
Where a function creates new propositional variables, these are given the name x_i where i is some identifier (typically a graph vertex). These variables are assumed to be unused in any other context.

2 Boolean Circuits

In contrast to the formulaic representation of propositional logic normally used, Boolean circuits are much closer to an electronics view of logic. Labelled input *wires* take the place of variables and together with (possibly unlabelled) internal wires they are connected by logic *gates* which compute various logic functions. This makes it very natural for the results of sub-circuits to be shared amongst other parts of the circuit, as would be expected in the physical world.

Boolean circuits may be efficiently represented as directed acyclic graphs (DAGs). Vertices having outgoing edges correspond to gates, with the edges pointing to the inputs to the gate. Vertices without outgoing edges (which we will call *leaf* vertices) are the input or output variables for the circuit.

Abdulla, Bjesse, and Eén proposed *reduced* Boolean circuits (RBCs) [1] as a DAG representation of a propositional formula with additional restrictions on the type and relationships of the gates which place RBCs somewhere between being a normal form and a canonical form for propositional formulæ. One of the key strengths of Boolean circuits is the ability to use one circuit to represent



(a) $a \wedge b \leftrightarrow (a \rightarrow b)$ reduced to $\neg(a \wedge b \leftrightarrow (a \wedge \neg b))$

(b) $a \vee (b \leftrightarrow \neg c)$ reduced to $\neg(\neg a \wedge (b \leftrightarrow c))$

Fig. 1. Example RBCs showing vertex labelling

a formula both positively and negatively. To preserve this property, Abdulla et al. eschew NNF in favour of restricting gates to conjunctions and equivalences (bi-implications), marking negation on the edges of the graph.

Definition 1 *An RBC is a DAG consisting of edges* $\mathbf{E}$ *and vertices* $\mathbf{V} = \mathbf{V_I} \cup \mathbf{V_L}$ *where internal vertices* $\mathbf{V_I}$ *represent operators, and leaf vertices* $\mathbf{V_L}$ *represent variables. The following properties are required to hold and form the encoding of Boolean circuits as DAGs:*

- *Each* $V \in \mathbf{V_I}$ *consists of an operator* $op(v) \in \{\wedge, \leftrightarrow\}$ *and a left and right edge* $(left(V), right(V) \in \mathbf{E})$*.*
- *Each* $V \in \mathbf{V_L}$ *contains a variable* $var(V)$*.*
- *Each* $E \in \mathbf{E}$ *has a sign* $sign(E) \in \{+, -\}$ *and a target vertex* $target(E) \in \mathbf{V}$*.*

The sign attribute encodes negation, where $sign(E) = +$ *indicates an unnegated edge and* $sign(E) = -$ *indicates a negated edge.*

The following additional properties serve to reduce the number of representations possible for equivalent formulæ:

- *All common subformulæ are shared:* $\forall V, V' \in \mathbf{V_I}, left(V) = left(V') \wedge right(V) = right(V') \rightarrow V = V'$*.*
- *The constant* $\top$ *only occurs in single-vertex RBCs.*
- *For all vertices,* $left(V) \neq right(V)$*.*
- *If* $op(V) = \leftrightarrow$ *then* $left(V)$ *and* $right(V)$ *are unsigned.*
- *There is a total order* $\prec$ *such that for all* $V \in \mathbf{V}$*,* $left(V) \prec right(V)$*.*

For example, Figure 1a shows the RBC representing the formula $a \wedge b \leftrightarrow \neg(a \rightarrow b)$, with some internal vertices annotated by a subscript capital. The annotations allow us to refer to the subformula $a \wedge b$ by the vertex A, for example, and also allows us to depict RBC fragments by identifying a vertex without giving any further details.

To simplify the definitions in this paper we extend the set of properties on RBC vertices and edges with the inverse functions of *target*, and *left* and *right*:

$$inedges(V) = \{E | E \in \mathbf{E}, target(E) = V\}$$

$$source(E) = \begin{cases} V & \text{if } E = left(V) \vee E = right(V) \\ \text{undefined} & \text{otherwise} \end{cases}$$

RBC operations Two RBCs rooted with edges L and R, may be composed given an operation $o \in \{\wedge, \leftrightarrow\}$ and a sign $s \in \{+, -\}$ to give the RBC $rbc(L, R, o, s)$ as follows:

- If o may be trivially evaluated using identity and other properties, return the result of doing so.
- Otherwise, check $L \prec R$ and swap if not.
- If $o = \leftrightarrow$ then s becomes $s \oplus sign(L) \oplus sign(R)$, and $sign(L)$ and $sign(R)$ become $+$ ($\oplus$ is the exclusive-or operation).
- The new vertex $V = \langle o, L, R \rangle$ is inserted into the DAG.
- The result is the edge $\langle sign, V \rangle$.

3 CNF Conversions on Linear Trees

We begin by examining CNF conversions for a restriction of RBCs, which will become a building block for the CNF conversions of full RBCs. Linear trees represent linear formulæ (those without equivalence operators) without taking into account the possibility for sharing.

Definition 2 *A* linear tree *is an RBC with the following changes to its structure:*

- *The only internal vertices are conjunction vertices*
- *No vertices are shared: the graph is a tree*

Given a linear tree, we define the following additional properties over vertices

$$inedge(V) = E \qquad \text{where } target(E) = V$$

$$sib(V) = \begin{cases} target(left(V')) & \text{if } inedge(V) = right(V') \\ target(right(V')) & \text{if } inedge(V) = left(V') \end{cases}$$

We give the various well-known CNF conversions informally and as depth-first procedures on linear trees. Each conversion produces a set of clauses which may be treated using the union ($\cup$) operator, combining two sets of clauses together, and the cross-multiply operator ($\times$), which forms the set of clauses corresponding to the disjunction of two sets, obtained by

$$a \times b = \{x \cup y | x \in a, y \in b\}$$

We use the notation $|C|$ to refer to the number of clauses in set C.

The *standard* CNF conversion is that obtained by exploiting the distributive properties of $\wedge$ and $\vee$ on a formula already in NNF to push disjunctions in towards the literals. This produces an equivalent (rather than equisatisfiable) formula at the expense of a potentially exponential number of clauses. Nevertheless, the conversion is optimal for some input formulæ. We define the conversion for linear trees as a recursive descent. CNF(T) given in Figure 2 denotes the standard CNF conversion of the subtree beginning at a root edge T.

$$\mathrm{CNF}(E) = \begin{cases} \mathrm{CNF}(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = + \\ \mathrm{CNF}^-(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = - \end{cases}$$

$$\mathrm{CNF}^-(E) = \begin{cases} \mathrm{CNF}^-(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = + \\ \mathrm{CNF}(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = - \end{cases}$$

$$\mathrm{CNF}(V) = \begin{cases} \mathit{var}(V) & \text{if } V \in \mathbf{V_L} \\ \mathrm{CNF}(\mathit{left}(V)) \cup \mathrm{CNF}(\mathit{right}(V)) & \text{if } \mathit{op}(V) = \wedge \end{cases}$$

$$\mathrm{CNF}^-(V) = \begin{cases} \neg\, \mathit{var}(V) & \text{if } V \in \mathbf{V_L} \\ \mathrm{CNF}^-(\mathit{left}(V)) \times \mathrm{CNF}^-(\mathit{right}(V)) & \text{if } \mathit{op}(V) = \wedge \end{cases}$$

Fig. 2. The standard clause form conversion for linear trees

3.1 Clause Form Conversions with Renaming

Renaming subformulæ is a strategy for reducing the number of clauses produced by a formula. The observation is made that a subformula may be replaced by a single variable if clauses are given to constrain that variable such that the satisfiability of the overall formula is unaffected. Such a conversion is said to be *equisatisfiable*: the introduced variables break equivalency. For example, the formula $(a \wedge b \wedge c) \vee (d \wedge e \wedge f)$ produces nine clauses in the standard conversion; introducing a new variable for the left-hand disjunct to produce the formula

$$x_{a \wedge b \wedge c} \vee (d \wedge e \wedge f) \quad \wedge \quad x_{a \wedge b \wedge c} \leftrightarrow (a \wedge b \wedge c)$$

with $x_{a \wedge b \wedge c}$ constrained by the equivalence on the right hand side results in only seven clauses. Nevertheless, it is satisfiable by precisely those assignments that satisfy the original formula.

The most straightforward algorithm of this type gives a new name to every internal vertex of the tree and is known as the *definitional* clause form conversion, given in Figure 3.

In fact, as observed by Plaisted and Greenbaum [10], if a subformula occurs with positive or negative polarity — if it appears under an even or odd number of negations — then only an implication is required to constrain the new variable, with the direction of the implication corresponding to the polarity of the subformula. We define the polarity function $\mathrm{pol}(T, V)$ for a vertex V in a linear trees T as

$$\mathrm{pol}(T, T) = 1$$

$$\mathrm{pol}(T, E) = \begin{cases} \mathrm{pol}(T, \mathit{source}(E)) & \text{if } \mathit{sign}(E) = + \\ -\,\mathrm{pol}(T, \mathit{source}(E)) & \text{if } \mathit{sign}(E) = - \end{cases}$$

$$\mathrm{pol}(T, V) = \mathrm{pol}(\mathit{inedge}(V))$$

$$\mathrm{DEF}(E) = \begin{cases} \mathrm{DEF}(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = + \\ \mathrm{DEF}^{-}(\mathit{target}(V)) & \text{if } \mathit{sign}(E) = - \end{cases}$$

$$\mathrm{DEF}(V) = \begin{cases} \mathit{var}(V) & \text{if } v \in \mathbf{V_L} \\ \{\{\neg x_V, x_{\mathit{target}(\mathit{left}(V))}\}, \{\neg x_V, x_{\mathit{target}(\mathit{right}(V))}\}\} & \\ \quad \cup \{\{x_V, \neg x_{\mathit{target}(\mathit{left}(V))}, \neg x_{\mathit{target}(\mathit{right}(V))}\}\} & \\ \quad \cup \mathrm{DEF}(\mathit{left}(V)) \cup \mathrm{DEF}(\mathit{right}(V)) & \text{if } \mathit{op}(V) = \wedge \end{cases}$$

$$\mathrm{DEF}^{-}(V) = \begin{cases} \neg\, \mathit{var}(V) & \text{if } v \in \mathbf{V_L} \\ \{\{x_V, x_{\mathit{target}(\mathit{left}(V))}\}, \{x_V, x_{\mathit{target}(\mathit{right}(V))}\}\} & \\ \quad \cup \{\neg x_V, \neg x_{\mathit{target}(\mathit{left}(V))}, \neg x_{\mathit{target}(\mathit{right}(V))}\}\} & \\ \quad \cup \mathrm{DEF}(\mathit{left}(V)) \cup \mathrm{DEF}(\mathit{right}(V)) & \text{if } \mathit{op}(V) = \wedge \end{cases}$$

Fig. 3. The definitional clause form conversion

In the example above, the subformula $a \wedge b \wedge c$ appears positively, so the renaming can be shortened to

$$x_{a \wedge b \wedge c} \vee (d \wedge e \wedge f) \quad \wedge \quad x_{a \wedge b \wedge c} \rightarrow (a \wedge b \wedge c)$$

producing only six clauses.

For linear trees, we consider only renamings of *vertices* (other analyses place an equivalent restriction on renaming subfomulæ with negation as the main connective). The order in which renamings are made does not affect the final result due to the commutivity of $\wedge$, so we are able to give renaming-based clause form conversions in terms of the sets of vertices that they rename. The transformation in Figure 4 constructs a graph consisting of the renamed formula and the subgraph defining constraints on the new variables. This is sufficient to allow us to write the structure-preserving clause form conversion due to Plaisted and Greenbaum [10] as

$$\mathrm{SP}(T) = \mathrm{CNF}(\mathrm{ren}(T, \mathbf{V_I}))$$

It is easy to construct cases where the definitional and structure-preserving conversions perform significantly worse than the standard conversion, despite the difference in asymptotic complexity. Consider, for example, the case of a formula already in conjunctive normal form. The structure-preserving conversion involves producing a new variable for each clause, with each definition taking one clause. The result is a worst-case doubling in the size of the clause form, where the standard conversion leaves the formula unchanged. A better approach is to construct the renaming sets more carefully, according to the overall impact that a renaming has.

3.2 The Conversion due to Boy de la Tour

Boy de la Tour [4] presents a comprehensive solution to the problem of choosing the subformulæ to rename. The approach taken is to compute the impact of

$$\mathrm{ren}(T,\mathbf{R}) = rbc(\mathrm{def}(T,T,\mathbf{R}), \mathrm{sub}(T,\mathbf{R}), \wedge, +)$$

$$\mathrm{def}(T,E,\mathbf{R}) = \mathrm{def}(T, target(E), \mathbf{R})$$

$$\mathrm{def}(T,V,\mathbf{R}) = \begin{cases} V & \text{if } V \in \mathbf{V_L} \\ rbc(\left\{ \begin{array}{ll} \top & \text{if } V \notin \mathbf{R} \\ rbc(x_V, \mathrm{sub}^-(V, \mathbf{R} \setminus \{V\}), \wedge, -) & \text{if } \mathrm{pol}(T,V) = 1 \\ rbc(\neg x_V, \mathrm{sub}^+(V, \mathbf{R} \setminus \{V\}), \wedge, -) & \text{if } \mathrm{pol}(T,V) = -1 \end{array} \right\}, & \\ \quad rbc(\mathrm{def}(T, left(V), \mathbf{R}), \mathrm{def}(T, right(V), \mathbf{R}), \wedge, +), & \\ \quad \wedge, +) & \text{if } V \in \mathbf{V_I} \end{cases}$$

$$\mathrm{sub}(E,\mathbf{R}) = \mathrm{sub}^{sign(E)}(target(E), \mathbf{R})$$

$$\mathrm{sub}^s(T,V,\mathbf{R}) = \begin{cases} V & \text{if } V \in \mathbf{V_L} \\ x_V & \text{if } V \in \mathbf{R} \\ rbc(\mathrm{sub}(left(V),\mathbf{R}), \mathrm{sub}(right(V),\mathbf{R}), op(V), s) & \text{otherwise} \end{cases}$$

Fig. 4. The vertex-based renaming construction $\mathrm{ren}(T, \mathbf{R})$. Function $\mathrm{sub}(T, \mathbf{R})$ returns the graph with root edge T with renamed subgraphs replaced by variables; $\mathrm{def}(T, T', \mathbf{R})$ returns the graph defining all the introduced variables below T' with respect to root T

Table 1. The clause counting functions $p^+(V)$ and $p^-(V)$

	$p^+(E)$	$p^-(E)$
$sign(E) = +$	$p^+(target(E))$	$p^-(target(E))$
$sign(E) = -$	$p^-(target(E))$	$p^+(target(E))$
	$p^+(V)$	$p^-(V)$
$v \in \mathbf{V_L}$	1	1
$op(V) = \wedge$	$p^+(left(V)) + p^+(right(V))$	$p^-(left(V))p^-(right(V))$

renaming any given subformula and to perform the renaming only if it will not increase the number of clauses produced by the formula as a whole. The conversion is shown to be optimal for formulæ without equivalences, and we will make use of this property in order to prove the optimality of the new conversion in Section 4.

Boy de la Tour defines the functions $p^+(T) = |\mathrm{CNF}(T)|$ and $p^-(T) = |\mathrm{CNF}(\neg T)|$ using a simple lookup table (Table 1) which enables these values to be computed without constructing the clauses themselves. The *benefit* (that is, the reduction in the total number of clauses) of renaming a vertex V in a tree T is given by

$$B(T,V) = p^+(T) - p^+(\mathrm{ren}(T, \{V\}))$$

In order to make a decision about renaming at a particular vertex without needing to analyse the whole tree, $p^+(T)$ is rewritten in terms of $p^+(V)$ and $p^-(V)$:

$$p^+(T) = a_V^T p^+(V) + b_V^T p^-(V) + c_V^T$$

Table 2. Computation of the coefficients a_V^T and b_V^T

	a_E^T	b_E^T
$E = T$	1	0
$sign(E) = +$	$a^T_{source(E)}$	$b^T_{source(E)}$
$sign(E) = -$	$b^T_{source(E)}$	$a^T_{source(E)}$
	a_V^T	b_V^T
$op(V) = \wedge$	$a^T_{inedge(V)}$	$b^T_{inedge(V)} p^-(sib\, V)$

$$\mathrm{BDLT}(T, E) = \mathrm{BDLT}(T, \mathit{target}(V))$$

$$\mathrm{BDLT}^+(T, V) = \begin{cases} \emptyset & \text{if } v \in \mathbf{V_L}, \text{ or} \\ \mathrm{BDLT}(T, \mathit{left}(V)) \cup \mathrm{BDLT}^+(T, \mathit{right}(V)) & \text{if } B(T, V) < 0, \text{ or} \\ \{V\} \cup \mathrm{BDLT}(\mathrm{ren}(T, \{V\}), \mathit{left}(v)) & \\ \quad \cup \mathrm{BDLT}(\mathrm{ren}(T, \{V\}), \mathit{right}(V)) & \text{if } B(T, V) \geq 0 \end{cases}$$

Fig. 5. Renaming sets construction for the Boy de la Tour conversion

Where the coefficients a, b may be considered as the number of occurrences of the clauses representing V and $\neg V$ respectively, such that the first sum counts the total number of clauses including subformulæ of V; the coefficient c represents the number of clauses due to the rest of the tree. a and b are computed from the context of V as in Table 2. Note that the values are related to the polarity of the vertices: $a_V^T = 0$ if $\mathrm{pol}(T, V) = -1$ and $b_V^T = 0$ if $\mathrm{pol}(T, V) = 1$. When computing the benefit, the coefficient c is cancelled, so we do not need to give its construction. The benefit function can now be given in terms of polarity as

$$\begin{array}{ll} a_V^T p^+(V) \; - \; (a_V^T + p^+(V)) & \text{if } \mathrm{pol}(T, V) = 1 \\ b_V^T p^-(V) \; - \; (b_V^T + p^-(V)) & \text{if } \mathrm{pol}(T, V) = -1 \end{array}$$

The algorithm given by Boy de la Tour is a top-down computation of the benefit of a renaming given the renamings that have gone before. We give the construction of the renaming set in Table 5 allowing us to write the algorithm as

$$\mathrm{BDLT}(T) = \mathrm{CNF}(\mathrm{ren}(T, \mathrm{BDLT}^+(T, T) \cup \mathrm{BDLT}^-(T, T)))$$

A dynamic programming implementation of $B(T, V)$ as given by Boy de la Tour [4] requires $O(1)$ computations at each vertex but the arithmetic is on $|\mathbf{V}|$-bit words which leads to a per-vertex complexity of $O(|\mathbf{V}|)$. The resulting algorithm is $O(|\mathbf{V}|^2)$ in contrast to DEF and SP which are both linear.

A more recent presentation of the algorithm by Nonnengart et al. [9] removes the requirement for arbitrary-length arithmetic by reducing $B(T, V) \geq 0$ to a number of case splits. Unfortunately, these can become quite elaborate: the conditions for zero polarity formulæ require the evaluation of eight syntactic conditions in various combinations.

	$p_r^+(E,\mathbf{R})$	$p_r^-(E,\mathbf{R})$
$sign(E) = +$	$p_r^+(target(E),\mathbf{R})$	$p_r^-(target(E),\mathbf{R})$
$sign(E) = -$	$p_r^-(target(E),\mathbf{R})$	$p_r^+(target(E),\mathbf{R})$

	$p_r^+(V,\mathbf{R})$	$p_r^-(V,\mathbf{R})$
$V \in \mathbf{V_L}$	1	1
$V \in \mathbf{R}$	1	1
$op(V) = \wedge$	$p_r^+(left(V),\mathbf{R}) + p_r^+(right(V),\mathbf{R})$	$p_r^-(left(V),\mathbf{R}) p_r^-(right(V),\mathbf{R})$

Table 3. The renaming-compensated clause counting functions $p_r^+(T,\mathbf{R})$ and $p_r^-(T,\mathbf{R})$

$$\mathrm{COMP}(T,E) = \mathrm{COMP}(T, target(V))$$

$$\mathrm{COMP}(T,V) = \begin{cases} \emptyset & \text{if } V \in \mathbf{V_L}\text{, or} \\ \mathrm{COMP}(T, left(V)) \cup \mathrm{COMP}(T, right(V)) & \text{if } \mathrm{pol}(T,V) = 1\text{, or} \\ \mathrm{dis}(V) \cup \mathrm{COMP}^-(T, left(V)) \cup \mathrm{COMP}^-(T, right(V)) & \text{if } \mathrm{pol}(T,V) = -1 \end{cases}$$

$$\mathrm{dis}(V) = \left\{ \begin{array}{ll} \emptyset & \text{if } n_l n_r \le n_l + n_r\text{, or} \\ \{left(V)\} & \text{if } n_l > n_r \\ \{right(V)\} & \text{if } n_l \le n_r \end{array} \right\} \text{ where } \left\{ \begin{array}{l} n_l = p_r^-(left(V), \mathrm{COMP}(T, left(V))) \\ n_r = p_r^-(right(V), \mathrm{COMP}(T, right(V))) \end{array} \right\}$$

Fig. 6. Renaming sets construction for the compact conversion

4 The Compact Conversion

We present a new clause form conversion, the *compact* conversion which computes the sets of renaming locally and bottom-up. For each vertex we consider the number of clauses it will generate based on whether a child vertex is renamed. Consider a disjunction $\phi \vee \psi$, with all subformulæ of ϕ and ψ already renamed as appropriate. The disjunction is converted by either renaming an argument, eg ϕ to x_ϕ, which produces a definition $x_\phi \rightarrow \phi$ and replaces the disjunction by the renamed form $x_\phi \vee \psi$; or alternatively computing $\mathrm{CNF}(\phi) \times \mathrm{CNF}(\psi)$ — the standard conversion of the disjunction. The decision is made based on which generates the most clauses, determined by the sum or the product, respectively, of the number of clauses in ϕ and ψ.

More precisely, we define the function $\mathrm{COMP}(T,V)$ in Figure 6 to give the set of renamings on the tree beginning at V. The auxiliary function $\mathrm{dis}(V)$ chooses the best child of V, if any, to rename by using the sum-versus-product decision. The renaming condition is computed on the tree after all vertices below the considered one have been renamed. To accommodate this we define a new pair of clause-counting functions $p_r^+(V,\mathbf{R})$ and $p_r^-(V,\mathbf{R})$ which count the number of clauses produced by the graph beginning at vertex V after the application of renaming $\mathbf{R}$ (Table 3). That is, $p_r^s(V,\mathbf{R}) = |\mathrm{sub}^s(V,\mathbf{R})|$ (the clauses in $\mathrm{def}^s(V,\mathbf{R})$ are disregarded as they play no further part in determining the size of the result).

Since we are targeting a SAT solver with this conversion, with its (assumed) exponential complexity in the number of variables, we choose to rename only if it *reduces* the number of clauses produced. In the case that the number of clauses is the same, the renaming is not performed. This is in contrast to the Boy de la Tour conversion, where the optimality analysis is simplified by the zero-benefit renaming.

5 Optimality of the Compact Conversion for Linear Trees

We show the optimality of the compact conversion by a comparison with the Boy de la Tour conversion. We establish which vertices appear in the renaming sets of one conversion and not the other, and then analyse the impact that the differences make.

When comparing the decision taken to include a vertex in the renaming sets by the two algorithms we take into account the different contexts: in the Boy de la Tour algorithm, the superformulæ have already been renamed; in the compact conversion the subformulæ have been renamed. Writing $\mathbf{R}$ for a set of renamings, we have $\mathbf{R}_{\sqsupset V}$ for the subset of renamings involving the superformulæ of V and $\mathbf{R}_{\sqsubset V}$ for the subset involving the subformulæ of V. The compact conversion depends only on p_r^+ and p_r^- but these are computed after subformula renaming. That is, the decision to rename the vertex V_1 in $V_1 \wedge V_2$ is based on the values $p_r^+(V_1, \mathbf{R}_{\sqsubset V_1})$, $p_r^-(V_2, \mathbf{R}_{\sqsubset V_2})$ and their complements. In contrast, for the Boy de la Tour algorithm the decision is based on the values $a_{V_1}^{\mathrm{ren}(T,\mathbf{R}_{\sqsupset V_1})}$, $b_{V_1}^{\mathrm{ren}(T,\mathbf{R}_{\sqsupset V_1})}$, $p^+(V_1)$, $p^-(V_1)$

We begin by establishing some basic lemmas about the Boy de la Tour coefficients and the clause counting functions.

Lemma 1. *For a vertex V and renaming $\mathbf{R}$ on tree T, $a_V^{\mathrm{ren}(T,\mathbf{R})} = 1$ if $\mathrm{pol}(T, V) = 1$, and $b_V^{\mathrm{ren}(T,\mathbf{R})} = 1$ if $\mathrm{pol}(T, V) = -1$*

Proof. After renaming, a vertex V becomes part of the definition of the replacement variable x_V. According to Figure 4, the definition is attached by a tree of positive conjunctions to the root with the sign of the inedge of V reflecting its original polarity. By the definition of a_V^T and b_V^T on conjunctions, the lemma holds.

Lemma 2. *For a vertex V and renamings $\mathbf{R}$ and $\mathbf{R}'$ with $\mathbf{R}' \subseteq \mathbf{R}$, $p_r^s(V, \mathbf{R}) \le p_r^s(V, \mathbf{R}') \le p^s(V)$*

Proof. This follows from the definitions of p_r^s and p_r. Both increase monotonically with tree depth. As renaming effectively prunes part of the tree, it can only reduce the values of the functions.

5.1 Positive Polarity

Lemma 3. *Neither conversion renames the children of positive polarity conjunctions. That is, for $\mathbf{pc} = \{V \in \mathbf{V}_I | \, \mathrm{pol}(T, source(inedge(V))) = 1\}$, $\mathbf{pc} \cap \mathrm{BDLT}(T, V) = \emptyset$ and $\mathbf{pc} \cap \mathrm{COMP}(T, V) = \emptyset$*

Proof. The argument for the compact conversion follows trivially from its definition. For the Boy de la Tour conversion, consider the vertex X in Figure 7a. The benefit of renaming, $B(T, X)$, is evaluated in the context $\text{ren}(T, \mathbf{R}_{\sqsupset X})$. From Figure 2, $a_X^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} = a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})}$, hence the benefit is

$$a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^+(X) \; - \; (a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} + p^+(X))$$

The condition $B(T, X) \geq 0$ reduces to $a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} \geq 2$ and $p^+(X) \geq 2$. From Lemma 1, in order to obtain the former vertex B must not be renamed. From $B \notin \mathbf{R}$, we deduce $\mathbf{R}_{\sqsupset B} = \mathbf{R}_{\sqsupset X}$ and hence write the condition $B(T, B) < 0$ as

$$a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^+(B) \; - \; (a_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} + p^+(B)) < 0$$

which together with the earlier conditions constrains $p^+(B) = 1$. Since B is a conjunction it produces $p^+(X) + p^+(Y)$ clauses and the condition on $p^+(X)$ is thus in conflict with the condition that B is not renamed.

The argument for Y follows similarly, as does the case of BX or BY being signed edges.

5.2 Negative Polarity

We break the negative polarity argument into several pieces, firstly simplifying the Boy de la Tour benefit function. Consider vertex X in Figure 7b. From Figure 2, $b_X^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} = b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^-(Y)$, hence the benefit of renaming $B(T, X)$, in the context $\text{ren}(T, \mathbf{R}_{\sqsupset X})$ is

$$b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^-(Y) p^+(X) \; - \; (b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^-(Y) + p^+(X))$$

We consider two cases for $B(T, X) \geq 0$. If $b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} = 1$ then the renaming decision is localised: it is based only on $p^+(X)$ and $p^-(Y)$:

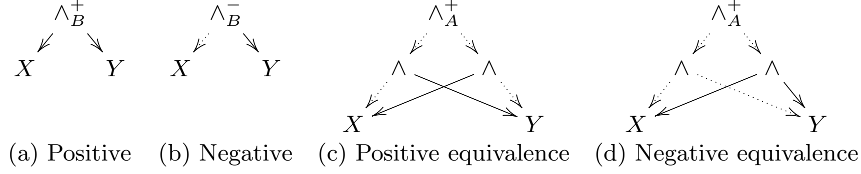
$$B'(T, X) = p^-(Y) p^+(X) - (p^-(Y) + p^+(X))$$

If $b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} \geq 2$, we must consider the same situation as for the positive case: the condition that $B \notin \mathbf{R}^-$. The inequality $B(T, B) < 0$ reduces to

$$b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})} p^-(B) < b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X}))} + p^-(B)$$

This holds only when $p^-(B) = 1$. Given $p^-(B) = p^+(X) p^-(Y)$ we also have $p^+(X) = p^-(Y) = 1$ and hence the vertex X is not renamed. This configuration is covered by the reduced condition $B'(T, X)$ which is thus sufficient condition for making the renaming decision. That is, the renaming decision is made independently of the value of $b_B^{\text{ren}(T,\mathbf{R}_{\sqsupset X})}$.

Lemma 4. *For linear trees, the renaming given by the Boy de la Tour algorithm with benefit function $B'(T, V)$ is the same as with the original function $B(T, V)$.*



(a) Positive (b) Negative (c) Positive equivalence (d) Negative equivalence

Fig. 7. RBC subgraphs for the optimality proofs and equivalence discussion

Proof. The argument for the children of negative polarity vertices is given above (the arguments for Y and different edge signs follow similarly). For children of positive polarity vertices, it is easy to see that Lemma 3 still holds. The remaining case is the root vertex, which is not renamed under either condition.

Using this reduced condition, the Boy de la Tour conversion has no restriction on the order of evaluation, which means that we can compare it more directly with the compact conversion. We define the conversion $\mathrm{BDLT}'(T, V)$ to be a bottom-up conversion using the benefit function $B'(T, V)$. From Lemmas 3 and 4 we know that $\mathrm{BDLT}(T, T) = \mathrm{BDLT}'(T, T)$ for all linear trees T. All remaining theorems are on this bottom-up conversion.

Lemma 5. *For all linear trees T,* $\mathrm{COMP}(T, T) \subseteq \mathrm{BDLT}'(T, T)$

Proof. We argue in the negative as it is more convenient. Consider vertex X in Figure 7a, with $X \notin \mathrm{BDLT}'(T, T)$. From the definition of the Boy de la Tour conversion, $B'(T, X) < 0$ which reduces to the two possibilities $p(X) = 1$ or $p(Y) = 1$. By Lemma 2, this means that either $p^+_r(X, \mathbf{R}_{\sqsubset B}) = 1$ or $p^+_r(Y, \mathbf{R}_{\sqsubset B}) = 1$ and hence the renaming condition for the compact conversion, $p^+_r(X, \mathbf{R}_{\sqsubset B}) p^+_r(Y, \mathbf{R}_{\sqsubset B}) > p^+_r(X, \mathbf{R}_{\sqsubset B}) + p^+_r(Y, \mathbf{R}_{\sqsubset B})$, is violated.

The argument follows similarly for Y.

Lemma 6. *For all linear trees T, with a renaming* $\mathbf{R} = \mathrm{COMP}(T, T)$*, for all* $V \notin \mathbf{R}$*,* $p^s_r(V, \mathbf{R}) = 1 \rightarrow p^s(V) = 1$

Proof. We show this by induction on the structure of the tree. The base case $V \in \mathbf{V_L}$ (V is a leaf) is trivial from the definition of p. For the step case, if V is a disjunction, then $p^s_r(\mathit{left}(V), \mathbf{R}) = p^s_r(\mathit{right}(V), \mathbf{R}) = 1$. This means, if $X = \mathit{target}(\mathit{left}(V))$ and $Y = \mathit{target}(\mathit{right}(V))$,

- $X \notin \mathbf{R}, Y \notin \mathbf{R}$: proof follows from the inductive hypothesis
- $X \notin \mathbf{R}, Y \in \mathbf{R}$: the condition necessary to rename Y is violated because, by Lemma 2, $p^s_r(X, \mathbf{R}_{\sqsubset V}) = p^s(X) = 1$.
- $X \in \mathbf{R}, Y \notin \mathbf{R}$: as above, by symmetry
- $X \in \mathbf{R}, Y \in \mathbf{R}$: prohibited by the definition of the compact conversion

V cannot be a conjunction as $p^s_r(V, \mathbf{R}) \geq 2$ is in contradiction with the induction hypothesis.

We can now fix the precise difference between the two conversions. Consider vertex X in Figure 7a, with $X \notin \mathrm{COMP}(T,T)$. By the definition of the compact conversion, $p_r^+(X, \mathbf{R}_{\sqsubset B}) p_r^+(Y, \mathbf{R}_{\sqsubset B}) \leq p_r^+(X, \mathbf{R}_{\sqsubset B}) + p_r^+(Y, \mathbf{R}_{\sqsubset B})$ which reduces to the three possibilities $p_r^+(X, \mathbf{R}_{\sqsubset B}) = 1$ or $p_r^+(Y, \mathbf{R}_{\sqsubset B}) = 1$ or $p_r^+(X, \mathbf{R}_{\sqsubset B}) = p_r^+(Y, \mathbf{R}_{\sqsubset B}) = 2$. In the first case, X may be a leaf vertex, in which case $X \notin \mathrm{BDLT}'(T,T)$, or a disjunction, in which case by Lemma 6, $p^+(X) = 1$ and hence[1] $X \notin \mathrm{BDLT}'(T,T)$. A conjunction is ruled out by the restriction on the number of clauses. The cases for Y and for signed edges follow similarly. For the final case, by Lemma 2, the Boy de la Tour conversion always renames either X or Y: this defines the set of vertices renamed by Boy de la Tour but not by compact.

Lemma 7. *For all linear trees T, $\mathrm{COMP}(T,T) \cup \mathbf{Z} = \mathrm{BDLT}'(T,T)$ where $\mathbf{Z}$ is the set of vertices such that for all $V \in \mathbf{Z}$, $p_r^+(V, \mathrm{COMP}(T,V)) = 2$ and $p_r^+(sib(V), \mathrm{COMP}(T,V)) = 2$*

Proof. From the discussion above and Lemma 5, no other vertex is in $\mathrm{BDLT}'(T,T)$ that is not in $\mathrm{COMP}(T,T)$.

Theorem 1. *The size of the clause form generated by the compact and Boy de la Tour conversions is the same: $p_r^+(T, \mathrm{COMP}(T,T)) = p_r^+(T, \mathrm{COMP}(T,T))$*

Proof. Since renamings may be applied in any order, we show that after applying those in $\mathrm{COMP}(T,T)$, the benefit of applying any of those in $\mathbf{Z}$ is zero. By Boy de la Tour's *fundamental theorem of monotonicity* [4], the members of $\mathbf{Z}$ may be considered in any order for this proof.

Consider a vertex $X \in \mathbf{Z}$ as depicted in Figure 7b. The benefit $B'(T,X)$ of renaming X after $\mathrm{COMP}(T,T)$ is $p_r^+(X, \mathrm{COMP}(T,T)) p_r^-(Y, \mathrm{COMP}(T,T)) - (p_r^+(X, \mathrm{COMP}(T,T)) + p_r^-(Y, \mathrm{COMP}(T,T))$. However, by the definition of $\mathbf{Z}$ in Lemma 7, and by Lemma 2, $p_r^+(X, \mathrm{COMP}(T,T)) = 2$ and $p_r^+(Y, \mathrm{COMP}(T,T)) = 2$, and hence $B'(T,V) = 0$.

6 Extension to RBCs

We have shown that the compact conversion produces an optimal number of clauses for linear trees, so we now extend the algorithm to general RBCs. The extension is heuristic: like Boy de la Tour, we do not claim optimality for the resulting clause form conversion.

Removal of Equivalences An RBC with equivalence vertices can be transformed into a linear RBC with only a linear increase in size by replacing equivalences with the subgraphs given in Figures 7c and d. The different treatments for positive and negative polarity equivalences reduce the number of clauses generated [10]. Note that a negative equivalence is replaced by a positive subgraph so the incoming edge must have its sign inverted.

[1] The case split for BDLT′ is given in the proof of Lemma 5

Polarity Zero Vertices The children of equivalence nodes are referenced both positively and negatively (as can be seen from the replacement subgraphs), sometimes referred to as *zero* polarity. Similarly, the sharing used in RBCs encourages a single vertex to be referenced with both polarities. We can convert an RBC with zero polarity vertices to one without by splitting every zero polarity vertex into a pair, one of each polarity, and suitably treating the incoming edges. Such treatment results in at most a doubling of the size of the RBC.

The substitution and subsequent splitting of equivalences differs significantly from the direct treatment of Boy de la Tour. In particular, Boy de la Tour's algorithm renames a descendant vertex of an equivalence both positively and negatively, simultaneously. This sometimes results in a tradeoff: the renaming of one polarity must have sufficient benefit to outweigh any negative benefit of renaming the other polarity. By splitting the polarities and treating them independently we improve the flexibility of the conversion and reduce the number of clauses in some circumstances, as compared to Boy de la Tour.

Shared Subgraphs Having removed equivalences and zero polarity vertices we are close to a linear tree structure. In fact, we can see the resulting structure as a collection of trees joined at the shared vertices. We can incorporate treatment of shared vertices into the bottom-up compact conversion algorithm by renaming any shared vertex which generates more than one clause and repeating the subgraph otherwise. The resulting algorithm is locally optimal as each constituent tree is optimally converted and the shared subgraphs are renamed only when renaming does not increase the resulting size.

7 Implementation and Evaluation

We have implemented the compact conversion extended to RBCs as part of the NuSMV model checker [5]. The implementation works directly on RBCs, performing the substitutions and duplications described above implicitly rather than constructing the resulting graph explicitly. Each vertex is considered as both a positive and a negative polarity vertex, and a depth-first traversal is used to mark each vertex with the number of incoming edges in each polarity. A second depth-first traversal produces the clause form directly. Bottom-up, each vertex is annotated with the clauses produced positively and negatively after renaming (ie, $\mathrm{CNF}(\mathrm{sub}(V, \mathrm{COMP}(T, V)))$), the definitional clauses being saved in a global variable (ie, $\mathrm{CNF}(\mathrm{def}(V, \mathrm{COMP}(T, V)))$). Whenever a shared vertex is encountered, it is renamed according to the strategy described above. No explicit computation of $p_r^s(V)$ is required: they correspond to the sizes of the sets of clauses — a constant time operation.

In Table 4 we compare the behaviour of the built-in CNF conversion in NuSMV (the definitional conversion) against the structure-preserving conversion and the compact conversion using two leading satisfiability solvers. The

Table 4. Benchmark results for three clause form conversions

Problem	Conv.	Clauses	Vars	Total literals	zChaff [7] Decisions	Time (s)	Jerusat [8] Time (s)
DME (Access)	Def	89150	31328	229882	40332	24.2	155.3
	SP	53285	22866	129840	39283	25.8	104.9
	Comp	22979	4986	70278	48232	10.6	32.1
DME (Priority)	Def	234515	79577	569387	28798	52.5	149.8
	SP	109637	51965	273339	21894	8.1	47.1
	Comp	52312	7587	456576	34936	5.2	3.53
DME (OT)	Def	737157	247079	1741365	25991	181.3	1084
	SP	280979	140302	700484	32023	50.4	150.9
	Comp	141604	12779	3322302	34808	10.4	38.9
Elevator	Def	234397	78483	548461	52450	68.3	369.2
	SP	109677	39373	274751	147791	74.4	338.3
	Comp	83901	23157	343673	168902	190	15.1

problems used are the standard DME benchmark[2] and a deadlock problem[3] (Elevator), as these were found to be representative of the behaviour on other hardware and deadlock problems. Unsurprisingly, the compact conversion consistently generates fewer clauses and the solving times are also better in most cases, sometimes dramatically so. More surprisingly, perhaps, is the increase in the number of decisions made by zChaff in every case: for the DME example, decisions are made more quickly, while for the Elevator, the rise in the number of decisions is more dramatic and the time taken for zChaff is increased. Interestingly, the time taken by Jerusat in this case is dramatically better than the best case for zChaff; it is otherwise usually outperformed by zChaff.

The results also illustrate the effect of the compact conversion preferring to repeat small sets of clauses rather than renaming them: the total number of literals is, in the worst case, double that for the definitional conversion; this is contrasted with the order of magnitude reductions in the number of variables!

8 Conclusions

Despite optimising a problem attribute that is not directly connected to the solving time — the number of clauses — the compact conversion algorithm produces a set of clauses that are in most cases more quickly solved. With the compact conversion, in contrast to the Boy de la Tour conversion, this is achieved without changing the complexity class of the conversion as compared to the more well-known clause form conversions.

[2] See [6] for more details

[3] Thanks to Toni Jussila for providing the files for this example

References

1. Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén. Symbolic reachability analysis based on SAT-solvers. In S. Graf and M. Schwartzbach, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS'00*, volume 1785 of *Lecture Notes in Computer Science*, pages 411–425. Springer-Verlag, March 2000.
2. Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Principles and Practice of Constraint Programming — 9th International Conference, CP 2003*, Lecture Notes in Computer Science, 2003.
3. Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In W.R. Cleaveland, editor, *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, July 1999.
4. Thierry Boy de la Tour. An optimality result for clause form translation. *Journal of Symbolic Computation*, 14:283–301, 1992.
5. A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings of the Eleventh Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499, Trento, Italy, July 1999. Springer-Verlag.
6. Alan Frisch, Daniel Sheridan, and Toby Walsh. A fixpoint based encoding for bounded model checking. In M D Aagaard and J W O'Leary, editors, *Formal Methods in Computer-Aided Design; 4th International Conference, FMCAD 2002*, volume 2517 of *Lecture Notes in Computer Science*, pages 238–254, Portland, OR, USA, November 2002. Springer-Verlag.
7. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, pages 530–535, Las Vegas, June 2001.
8. Alexander Nadel. Backtrack search algorithms for propositional logic satisfiability: Review and innovations. Master's thesis, Tel-Aviv University, November 2002.
9. Andreas Nonnengart, Georg Rock, and Christoph Weidenbach. On generating small clause normal forms. In Claude Kirchner and Hélène Kirchner, editors, *Fifteenth International Conference on Automated Deduction*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 397–411. Springer-Verlag, 1998.
10. David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, September 1986.
11. M. N. Velev. Efficient translation of Boolean formulas to CNF in formal verification of microprocessors. In *Asia and South Pacific Design Automation Convference (ASP-DAC '04)*, January 2004.