

The Optimality of a Fast CNF Conversion and its Use with SAT

Paul Jackson and Daniel Sheridan¹

Abstract. Despite the widespread use and study of Boolean satisfiability for a diverse range of problem domains, encoding of problems is usually given to general propositional logic with little or no discussion of the conversion to clause form that will be necessary. In this paper we present a fast and easy to implement conversion to equisatisfiable clause form for Boolean circuits, a popular representation of propositional logic formulae. We show that the conversion is equivalent to that of Boy de la Tour and is hence optimal in the number of clauses produced for linear input formulae (formulae excluding \leftrightarrow), and we discuss the optimality for other input formulae.

We present experimental results for this and other conversion procedures on BMC problems demonstrating its superiority, and conclude that the CNF conversion plays a large part in reducing the overall solving time.

1 Introduction

Boolean satisfiability procedures have been applied to a wide variety of problems, most notably in planning [13] and model checking [4]. Model checking in particular has been a major drive in increasing the size of formulae that can be handled by SAT technology. The SAT solvers based on the DPLL procedure [7] which are typically used in solving these large problems require their input to be in conjunctive normal form (CNF), also called clause form. Earlier papers dealing with encoding to SAT, particularly much of the planning literature, encode directly from the input representation to clause form. Similarly, encodings from constraint satisfaction problems to SAT are given directly to clause form, allowing the authors to make comparisons between the algorithms used in the two domains [10].

More recent encoding work makes little mention of CNF conversion. Biere et al., proposing BMC [4], give an encoding to propositional logic only. Later work concentrating on the behaviour of the SAT solver on BMC problems [18] begins its analysis with the output of the solver. Similarly, although the SNF encoding for BMC [9] discusses the clauses generated, the majority of the presentation is in general propositional logic. The microprocessor verification work of Velev includes a thorough analysis of improving the clause form generated [20], 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 [3] 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. Reformulating CNF problems automatically using a variety of restricted resolution procedures has been extensively studied [8], but the procedures are frequently slow. By generating the improved clause form directly from the original problem, this may be avoided.

In the first-order logic domain, the CNF conversion problem was handled comprehensively by Boy de la Tour [5]. The algorithm given is impractical without the improvements by Nonnengart et al. [15], but the resulting algorithm is very fiddly to implement making it hard to be confident of a correct implementation.

In this paper we introduce a simple CNF conversion algorithm for propositional logic and prove its optimality with respect to the number of clauses. The existing optimal algorithm [5] is used to justify each decision made by the new algorithm, which allows us to isolate the cases in which the algorithms behave differently. We also present experimental results from NuSMV comparing the new algorithm with the more well known algorithms including the structure-preserving conversion [16] and that used in BCZChaff [11].

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* representing 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 descendants 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.

¹ University of Edinburgh, Kings Buildings, UK
email:d.j.sheridan@sms.ed.ac.uk

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. Since edges correspond to wires the semantics of the circuit means that a formula is referred to by an edge (and the graph below it) and implicitly the graph below it. This also means that the graphs have a somewhat non-standard structure including edges with no ancestor vertex.

Abdulla, Bjesse, and Eén proposed *reduced* Boolean circuits (RBCs) [1] as a similar 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 formulae. One of the key strengths of Boolean circuits is the ability to use one circuit to represent 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. An ordering relation is placed on RBCs effectively removing the commutivity of the gates. While this is not specified fully in [1], implementations can use the memory addresses of vertices to achieve the necessary total order on RBCs. This means that two RBCs constructed independently from similar formulae will have identical structures down to the ordering of vertices attached to vertices.

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 representing operators, and leaf vertices \mathbf{V}_L representing 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. Clearly, for full expressivity a formula in an RBC begins with an edge.

An RBC has the following additional properties which serve to reduce the number of representations possible for equivalent formulae:

- All common subformulae 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, $left(V) \neq right(V)$.

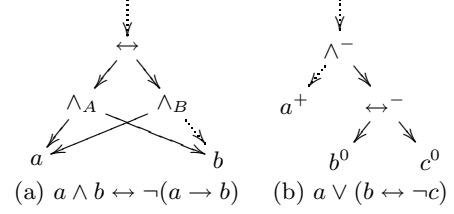


Figure 1. Example RBCs showing vertex labelling

- If $op(V) = \leftrightarrow$ then $left(V)$ and $right(V)$ are unsigned.
- There is a total order \prec on RBCs such that for all vertices $left(V) \prec right(V)$.

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*:

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

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

Where we restrict the structure of RBCs to trees (as for much of this paper) a vertex V has $|inedges(V)| = 1$ so we give a separate function for this case; it also makes sense to talk about the sibling of a vertex: the only other vertex with the same ancestor.

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

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

2.0.1 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 $rbcomp(L, R, o, s)$ 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 $+$.
- The new vertex $V = \langle o, L, R \rangle$ is inserted into the DAG.
- The result is the edge $\langle sign, V \rangle$.

The eventual role of a conjunction vertex (as a conjunction or a disjunction) is determined by the number of negated edges appearing on a path connecting it to the root of the graph. In a manner similar to polarity for propositional logic we define in Figure 2.0.1 the number of *positive references* at a vertex V as the number of incoming edges from a positive

polarity ancestor, and the number of *negative references* as the number of incoming edges from a negative polarity ancestor. These functions are defined with respect to the root edge of a formula so that only relevant edges are considered.

That is, $r_T^+(V)$ is increased by one for every positive incoming reference, but both positive and negative occurrences of equivalence reference their child vertices independently and hence increase the number of positive references separately (this can be motivated by the transformations in Section 5.2); similarly for r_T^- . The conventional notion of polarity in propositional formulæ does not completely capture the semantics of r_T^+ and r_T^- even on RBCs structured as trees; nevertheless, we will use the following terms as a shorthand:

- V has *positive polarity* if $r_T^+(V) \geq 1, r_T^-(V) = 0$
- V has *negative polarity* if $r_T^+(V) = 0, r_T^-(V) \geq 1$
- V has *zero polarity* if $r_T^+(V) \geq 1, r_T^-(V) \geq 1$

We use a system of annotations to the RBCs to indicate the polarity of a vertex, allowing us to draw diagrams of RBC fragments with properties that depend on the context of the fragment. Figure 1b shows the RBC corresponding to the formula $a \vee (b \leftrightarrow \neg c)$.

For the purposes of this paper we will be working with RBCs. The restriction on operators has the advantage of simplifying the algorithms and their analysis. To begin with we will simplify the situation even further by disallowing sharing in RBCs, treating them as trees. We add the handling of shared subformulæ in Section 5.3.

3 CNF Conversions

We give the various well-known CNF conversions informally and as depth-first procedures on RBCs. 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

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

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

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 RBCs as a recursive descent of the graph. $CNF(T)$ given in Figure 3 denotes the standard CNF conversion of the subgraph beginning at a root edge T .

3.1 Clause Form Conversions with Renaming

Renaming subformulæ is an established 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 variable breaks equivalency. For example, the formula $(a \wedge b \wedge c) \vee (d \wedge e \wedge f)$ produces nine clauses in the naïve conversion, but 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 the new variable $x_{a \wedge b \wedge c}$ 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. In fact, as observed by Plaisted and Greenbaum [16], if a subformula occurs with positive or negative polarity (not below an equivalence), then only an implication is required to constrain the new variable, with the direction of the implication corresponding to the polarity of the subformula. The process may be seen as the inverse of resolution. Indeed, the original clauses can be recovered by binary resolution on the introduced variable.

For RBCs, we consider only renamings of *vertices* (other analyses place an equivalent restriction on renaming subformulæ 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 RBC 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 define the structure-preserving clause form conversion due to Plaisted and Greenbaum [16] as

$$SP(T) = CNF(\text{ren}(T, \mathbf{V}_I))$$

For comparison, we give the explicit algorithm in Figure 5. However the function given does not allow us to write down the simpler clause form conversion (given explicitly in Figure 6) used in NuSMV [6] and BCZChaff [11] which does not distinguish between subformulæ of different polarities, and it is not sufficiently flexible to define the new conversion which we will be presenting in Section 4. We broaden the definition of renaming by using two sets: the set of vertices to be positively renamed, \mathbf{R}^+ , and the set to be negatively renamed, \mathbf{R}^- . Since $a \rightarrow b \wedge b \rightarrow a \equiv a \leftrightarrow b$, zero-polarity renamings are represented by a vertex's membership of both sets. This turns out to be difficult to represent as a graph transformation: if a single vertex is renamed in only one polarity then two distinct vertices must be given in the transformed graph. Since equivalences reference their children both positively and negatively, they must be replaced by their definitions in terms of conjunctions (see Section 5.2). We give instead a direct conversion to clause form in Figure 7 which allows us to define $DEF(T)$ and $SP(T)$ as

$$DEF(T) = CNF_R(T, \mathbf{V}_I, \mathbf{V}_I)$$

$$SP(T) = CNF_R(T, \{V \mid V \in \mathbf{V}_I, r_T^+(V) \geq 1\}, \{V \mid V \in \mathbf{V}_I, r_T^-(V) \geq 1\})$$

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

$$\begin{aligned}
r_T^+(V) &= \sum_{E \in \text{inedges}(V)} \begin{cases} 1 & \text{if } E = T, \text{sign}(E) = + \\ \min(r_T^+(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \wedge, \text{sign}(E) = + \\ \min(r_T^-(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \wedge, \text{sign}(E) = - \\ \min(r_T^+(\text{source}(E)), 1) + \min(r_T^-(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \leftrightarrow \\ 0 & \text{otherwise} \end{cases} \\
r_T^-(V) &= \sum_{E \in \text{inedges}(V)} \begin{cases} 1 & \text{if } E = T, \text{sign}(E) = - \\ \min(r_T^-(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \wedge, \text{sign}(E) = + \\ \min(r_T^+(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \wedge, \text{sign}(E) = - \\ \min(r_T^+(\text{source}(E)), 1) + \min(r_T^-(\text{source}(E)), 1) & \text{if } \text{op}(\text{source}(E)) = \leftrightarrow \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

Figure 2. The positive and negative references of vertex V with respect to root edge T

$$\begin{aligned}
\text{CNF}(E) &= \begin{cases} \text{CNF}(\text{target}(V)) & \text{if } \text{sign}(E) = + \\ \text{CNF}^-(\text{target}(V)) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{CNF}^-(E) &= \begin{cases} \text{CNF}^-(\text{target}(V)) & \text{if } \text{sign}(E) = + \\ \text{CNF}(\text{target}(V)) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{CNF}(V) &= \begin{cases} \text{var}(v) & \text{if } v \in \mathbf{V}_L \\ \text{CNF}(\text{left}(V)) \cup \text{CNF}(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ (\text{CNF}(\text{left}(V)) \times \text{CNF}^-(\text{right}(V))) \cup (\text{CNF}^-(\text{left}(V)) \times \text{CNF}(\text{right}(V))) & \text{if } \text{op}(V) = \leftrightarrow \end{cases} \\
\text{CNF}^-(V) &= \begin{cases} \text{CNF}^-(\text{left}(V)) \times \text{CNF}^-(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ \neg \text{var}(v) & \text{if } v \in \mathbf{V}_L \\ (\text{CNF}(\text{left}(V)) \times \text{CNF}(\text{right}(V))) \cup (\text{CNF}^-(\text{left}(V)) \times \text{CNF}^-(\text{right}(V))) & \text{if } \text{op}(V) = \leftrightarrow \end{cases}
\end{aligned}$$

Figure 3. The standard clause form conversion

$$\begin{aligned}
\text{ren}(T, \mathbf{R}) &= \text{rbccomp}(\text{def}(T, T, \mathbf{R}), \text{sub}(T, \mathbf{R}), \wedge, +) \\
\text{def}(T, E, \mathbf{R}) &= \text{def}(T, \text{target}(E), \mathbf{R}) \\
\text{def}(T, V, \mathbf{R}) &= \begin{cases} V & \text{if } V \in \mathbf{V}_L \\ \text{rbccomp} \left(\begin{cases} \top & \text{if } V \notin \mathbf{R} \\ \text{rbccomp}(x_V, \text{sub}^-(V, \mathbf{R}), \leftrightarrow, +) & \text{if } r_T^+(V) > 0 \text{ and } r_T^-(V) > 0, \text{ or} \\ \text{rbccomp}(x_V, \text{sub}^-(V, \mathbf{R}), \wedge, -) & \text{if } r_T^+(V) = 0, \text{ or} \\ \text{rbccomp}(\neg x_V, \text{sub}^+(V, \mathbf{R}), \wedge, -) & \text{if } r_T^-(V) = 0 \end{cases} \right), & \\ \text{rbccomp}(\text{def}(T, \text{left}(V), \mathbf{R}), \text{def}(T, \text{right}(V), \mathbf{R}), \wedge, +), & \\ \wedge, +) & \text{if } V \in \mathbf{V}_I \end{cases} \\
\text{sub}(E, \mathbf{R}) &= \text{sub}^{\text{sign}(E)}(\text{target}(E), \mathbf{R}) \\
\text{sub}^s(T, V, \mathbf{R}) &= \begin{cases} V & \text{if } V \in \mathbf{V}_L \\ x_V & \text{if } V \in \mathbf{R} \\ \text{rbccomp}(\text{sub}(\text{left}(V), \mathbf{R}), \text{sub}(\text{right}(V), \mathbf{R}), \text{op}(V), s) & \text{otherwise} \end{cases}
\end{aligned}$$

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

$$\begin{aligned}
\text{SP}(E) &= \begin{cases} \text{SP}(\text{target}(V)) & \text{if } \text{sign}(E) = + \\ \text{SP}^-(\text{target}(V)) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{SP}^-(E) &= \begin{cases} \text{SP}^-(\text{target}(V)) & \text{if } \text{sign}(E) = + \\ \text{SP}(\text{target}(V)) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{SP}(V) &= \begin{cases} \text{var}(V) & \text{if } v \in \mathbf{V}_L \\ \{\{\neg x_V, x_{\text{target}(\text{left}(V))}\}, \{\neg x_V, x_{\text{target}(\text{right}(V))}\}\} \cup \text{SP}(\text{left}(V)) \cup \text{SP}(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ \{\{\neg x_V, x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}, \{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}\} & \text{if } \text{op}(V) = \leftrightarrow \\ \cup \text{SP}(\text{left}(V)) \cup \text{SP}(\text{right}(V)) \cup \text{SP}^-(\text{left}(V)) \cup \text{SP}^-(\text{right}(V)) & \end{cases} \\
\text{SP}^-(V) &= \begin{cases} \neg \text{var}(V) & \text{if } v \in \mathbf{V}_L \\ \{\{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} \cup \text{SP}(\text{left}(V)) \cup \text{SP}(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ \{\{\neg x_V, x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}, \{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} & \text{if } \text{op}(V) = \leftrightarrow \\ \cup \text{SP}(\text{left}(V)) \cup \text{SP}(\text{right}(V)) \cup \text{SP}^-(\text{left}(V)) \cup \text{SP}^-(\text{right}(V)) & \end{cases}
\end{aligned}$$

Figure 5. The structure-preserving clause form conversion

$$\begin{aligned}
\text{DEF}(E) &= \begin{cases} \text{DEF}(\text{target}(V)) & \text{if } \text{sign}(E) = + \\ \text{DEF}^-(\text{target}(V)) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{DEF}(V) &= \begin{cases} \text{var}(V) & \text{if } v \in \mathbf{V}_L \\ \{\{\neg x_V, x_{\text{target}(\text{left}(V))}\}, \{\neg x_V, x_{\text{target}(\text{right}(V))}\}, \{x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} \\ \cup \text{DEF}(\text{left}(V)) \cup \text{DEF}(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ \{\{\neg x_V, x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}, \{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}\} \\ \cup \{\{x_V, x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}, \{x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} \\ \cup \text{DEF}(\text{left}(V)) \cup \text{DEF}(\text{right}(V)) & \text{if } \text{op}(V) = \leftrightarrow \end{cases} \\
\text{DEF}^-(V) &= \begin{cases} \neg \text{var}(V) & \text{if } v \in \mathbf{V}_L \\ \{\{x_V, x_{\text{target}(\text{left}(V))}\}, \{x_V, x_{\text{target}(\text{right}(V))}\}, \{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} \\ \cup \text{DEF}(\text{left}(V)) \cup \text{DEF}(\text{right}(V)) & \text{if } \text{op}(V) = \wedge \\ \{\{x_V, x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}, \{x_V, \neg x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}\} \\ \cup \{\{\neg x_V, x_{\text{target}(\text{left}(V))}, x_{\text{target}(\text{right}(V))}\}, \{\neg x_V, \neg x_{\text{target}(\text{left}(V))}, \neg x_{\text{target}(\text{right}(V))}\}\} \\ \cup \text{DEF}(\text{left}(V)) \cup \text{DEF}(\text{right}(V)) & \text{if } \text{op}(V) = \leftrightarrow \end{cases}
\end{aligned}$$

Figure 6. The definitional clause form conversion

$$\begin{aligned}
\text{CNF}_R(T, \mathbf{R}^+, \mathbf{R}^-) &= \text{def}(T, T, \mathbf{R}^+, \mathbf{R}^-) \cup \text{sub}(T, \mathbf{R}^+, \mathbf{R}^-) \\
\text{def}(T, E, \mathbf{R}^+, \mathbf{R}^-) &= \text{def}(T, \text{target}(E), \mathbf{R}^+, \mathbf{R}^-) \\
\text{def}(T, V, \mathbf{R}^+, \mathbf{R}^-) &= \begin{cases} \{\{\neg x_E\} \times \text{sub}(E, \mathbf{R}^+, \mathbf{R}^-) & \text{if } r_T^+ > 0 \\ \emptyset & \text{if } r_T^+ = 0 \end{cases} \cup \begin{cases} \{x_E\} \times \text{CNF}_R^-(E, \mathbf{R}^+, \mathbf{R}^-) & \text{if } r_T^- > 0 \\ \emptyset & \text{if } r_T^- = 0 \end{cases} \\
&\cup \text{def}(T, \text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \cup \text{def}(T, \text{right}(V), \mathbf{R}^+, \mathbf{R}^-) \\
\text{sub}^+(E, \mathbf{R}^+, \mathbf{R}^-) &= \begin{cases} \text{sub}^+(\text{target}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{sign}(E) = + \\ \text{sub}^-(\text{target}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{sub}^-(E, \mathbf{R}^+, \mathbf{R}^-) &= \begin{cases} \text{sub}^-(\text{target}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{sign}(E) = + \\ \text{sub}^+(\text{target}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{sign}(E) = - \end{cases} \\
\text{sub}^+(V, \mathbf{R}^+, \mathbf{R}^-) &= \begin{cases} \text{var}(v) & \text{if } v \in \mathbf{V}_L \\ x_V & \text{if } V \in \mathbf{R}^+ \\ \text{sub}^+(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \cup \text{sub}^+(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{op}(V) = \wedge \\ (\text{sub}^+(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \times \text{sub}^-(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-)) \\ \cup (\text{sub}^-(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \times \text{sub}^+(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-)) & \text{if } \text{op}(V) = \leftrightarrow \end{cases} \\
\text{sub}^-(V, \mathbf{R}^+, \mathbf{R}^-) &= \begin{cases} \neg \text{var}(v) & \text{if } v \in \mathbf{V}_L \\ \neg x_V & \text{if } V \in \mathbf{R}^- \\ \text{sub}^-(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \times \text{sub}^-(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-) & \text{if } \text{op}(V) = \wedge \\ (\text{sub}^+(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \times \text{sub}^+(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-)) \\ \cup (\text{sub}^-(\text{left}(V), \mathbf{R}^+, \mathbf{R}^-) \times \text{sub}^-(\text{right}(V), \mathbf{R}^+, \mathbf{R}^-)) & \text{if } \text{op}(V) = \leftrightarrow \end{cases}
\end{aligned}$$

Figure 7. The general renaming clause form conversion

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. There are a number of ways to overcome this. Small optimisations such as not renaming the children of positive vertices, or combining negative conjunction vertices together to form longer clauses, are easy to implement but do not make much impact overall. 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

In [5], Boy de la Tour presents a comprehensive solution to the problem of choosing the subformulae to rename. The approach taken is to compute the impact of renaming any given subformula and to perform the renaming if it will not increase the number of clauses produced by the formula as a whole. The conversion is shown to be optimal for formulae without equivalences, and we will make use of this property in order to prove the optimality of the new conversion in Section 4. The adaptation to RBC trees presented below removes some of the ambiguities from the original presentation as we can refer uniquely to vertices in the tree rather than to subformulae.

Boy de la Tour defines the functions $p^+(T) = |\text{CNF}(T)|$ and $p^-(T) = |\text{CNF}(\neg T)|$ 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

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

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 $p^-(V)$:

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

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 subformulae of V ; the coefficient c represents the number of clauses due to the rest of the tree. The coefficients a and b are computed from the context of V as in Table 2. Note that the coefficients are related to the polarity of the vertices: $a_V^T = 0$ if $r_T^-(V) = 0$ and $b_V^T = 0$ if $r_T^+(V) = 0$. When computing the benefit, the coefficient c cancels out, so we do not need to give its construction. The benefit function can now be given in terms of polarity as

$$\begin{array}{ll} a_V^T p^+(V) - (a_V^T + p^+(V)) & \text{if } r_T^-(V) = 0 \\ b_V^T p^-(V) - (b_V^T + p^-(V)) & \text{if } r_T^+(V) = 0 \\ a_V^T p^+(V) + b_V^T p^-(V) & \text{otherwise} \\ - (a_V^T + b_V^T + p^+(V) + p^-(V)) & \end{array}$$

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. Vertices are renamed according to their polarity so a single renaming set is sufficient and the renaming function of Figure 4 may be used. Nevertheless, we give

the construction of the positive and negative renaming sets BDLT^+ and BDLT^- in Table 8 as this will simplify the analysis later in the paper. The conversion algorithm may thus be written in two ways:

$$\begin{aligned} \text{BDLT}(T) &= \text{CNF}(\text{ren}(T, \text{BDLT}^+(T, T) \cup \text{BDLT}^-(T, T))) \\ \text{BDLT}(T) &= \text{CNF}_R(T, \text{BDLT}^+(T, T), \text{BDLT}^-(T, T)) \end{aligned}$$

Computing $B(T, V)$ is $O(|\mathbf{V}|)$, so the resulting algorithm is $O(|\mathbf{V}|^2)$ in contrast to DEF and SP which are both linear.

The main drawback of this approach is that the value of the functions p^+ and p^- can grow exponentially with the depth of the tree; precomputing the values as suggested by Boy de la Tour is likely to be unimplementable because of the large numbers involved: examining these functions in relation to BMC [17] demonstrates that even trivial examples will overflow a 32-bit register. A more recent presentation of the algorithm by Nonnengart et al. [15] does not change the asymptotic complexity but reduces the inequality $B(T, V) \geq 0$ to a number of case splits. For example, if the polarity of V is 1, it is renamed if $p^+(V) > 1$ and $a_V^T > 1$; $p^+(V) > 1$ can be checked by searching V for equivalences or conjunctions with positive polarity and similar conditions exist for a_V^T . Unfortunately, these can become quite elaborate. From the source code to SPASS [19], we see that computing the condition for polarity 0 formulae requires the evaluation of eight distinct syntactic conditions in various combinations. We argue that despite the theoretical advantages of this method, it is prohibitively difficult to ensure a correct implementation.

4 The Compact Conversion

We present a new clause form conversion, the *compact* conversion² which computes the sets of renaming locally and bottom-up. Intuitively, we precompute r_T^+ and r_T^- for each vertex, then beginning with the leaves we work upwards through the graph computing the clauses which represent each vertex. A separate set of definitional clauses is maintained to define the variables used for renaming. At each vertex we consider the number of clauses it will generate based on whether a child is renamed: a disjunction $x \vee y$ is converted by either renaming an argument x and producing the clauses $\{\neg x\} \times \text{CNF}(y)$ or by simply computing $\text{CNF}(x) \times \text{CNF}(y)$, whichever results in the fewest clauses. Equivalences are handled as conjunctions of disjunctions so the same test can be used. Note that the decision to rename a vertex is made when considering the clauses generated by its *parent* vertex. The clauses for the positive and/or negative cases as required are then easily generated.

More precisely, we define the functions $\text{COMP}^+(T, V)$ and $\text{COMP}^-(T, V)$ in Figure 9 to give the set of positive and negative renamings respectively on the graph beginning at V . The auxiliary function $\text{dis}^{xy}(V)$ chooses the best child of V , if any, to rename given their signs, x and y . 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

² The name is chosen to indicate that the conversion is compact both in its *output* (number of clauses with respect to the standard and structure-preserving conversions) and its *implementation* (with respect to the Nonnengart et al. implementation of Boy de la Tour conversion).

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

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

	a_E^T	b_E^T
$E = T$	1	0
$sign(E) = +$	$a_{source(E)}^T$	$b_{source(E)}^T$
$sign(E) = -$	$b_{source(E)}^T$	$a_{source(E)}^T$
	a_V^T	b_V^T
$op(V) = \wedge$	$a_{inedge(V)}^T$	$b_{inedge(V)}^T p^-(sib(V))$
$op(V) = \leftrightarrow$	$a_{inedge(V)}^T p^-(sib(V)) + b_{inedge(V)}^T p^+(sib(V))$	$a_{inedge(V)}^T p^+(sib(V)) + b_{inedge(V)}^T p^-(sib(V))$

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

$$\text{BDLT}^+(T, E) = \text{BDLT}(T, target(V))$$

$$\text{BDLT}^-(T, E) = \text{BDLT}^-(T, target(V))$$

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

$$\text{BDLT}^-(T, V) = \begin{cases} \emptyset & \text{if } V \in \mathbf{V}_L, \text{ or} \\ \text{BDLT}^-(T, left(V)) \cup \text{BDLT}^-(T, right(V)) & \text{if } r_T^-(V) = 0, \text{ or} \\ \text{BDLT}^-(T, left(V)) \cup \text{BDLT}^-(T, right(V)) & \text{if } B(T, V) < 0, \text{ or} \\ V \cup \text{BDLT}^-(\text{ren}(T, \{V\}), left(V)) \cup \text{BDLT}^-(\text{ren}(T, \{V\}), right(V)) & \text{if } B(T, V) \geq 0 \end{cases}$$

Figure 8. Renaming sets construction for the Boy de la Tour conversion

$p_r^+(V, \mathbf{R}^+, \mathbf{R}^-)$ and $p_r^-(V, \mathbf{R}^+, \mathbf{R}^-)$ which count the number of clauses produced by the graph beginning at vertex V after the renaming \mathbf{R} has been applied (see Table 3). That is, $p_r^s(V, \mathbf{R}^+, \mathbf{R}^-) = |\text{sub}^s(V, \mathbf{R}^+, \mathbf{R}^-)|$. Clearly, if the conversion to CNF is integrated with the computation of the renaming sets (as described above) then these functions are simply the sizes of the clause sets at V . In this way, the conversion takes just $O(\mathbf{V})$ time.

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 [5] is simplified by the zero-benefit renaming.

5 Optimality of the Compact Conversion

We are presenting the compact conversion as a simpler replacement for the Boy de la Tour conversion, and as such we must show that its performance is no worse. In fact, the Boy de la Tour conversion is known to be optimal for formulæ without equivalence operators [5], so we show that in this situation compact produces the same number of clauses. As noted above, we initially assume that the RBC is structured as a tree — no subformulæ are shared — to simplify the analysis.

We modify the renaming condition in the Boy de la Tour conversion to be strict ($B(T, V) > 0$), in accordance with the aims of the compact conversion. Boy de la Tour states [5] that this does not increase the number of clauses, so the comparison remains valid.

5.1 Optimality Without Equivalences

When comparing the decision taken to include a vertex in the renaming sets by the two algorithms, we must 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}^s for a set of renamings, $s \in \{+, -\}$, we have $\mathbf{R}_{\sqsupset V}^s$ for the subset of renamings involving the superformulæ of V and $\mathbf{R}_{\sqsubseteq V}^s$ for the subset involving the subformulæ of V , we can clarify the dependencies of the conversions. 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 $p_r^+(V_1, \mathbf{R}_{\sqsubseteq V_1}^+, \mathbf{R}_{\sqsubseteq V_1}^-)$, $p_r^+(V_2, \mathbf{R}_{\sqsubseteq V_2}^+, \mathbf{R}_{\sqsubseteq V_2}^-)$ and their complements. In contrast, for the Boy de la Tour algorithm the decision is based on the values $a_{V_1}^{\text{ren}(T, \mathbf{R}_{\sqsupset V_1}^+ \cup \mathbf{R}_{\sqsupset V_1}^-)}$, $b_{V_1}^{\text{ren}(T, \mathbf{R}_{\sqsupset V_1}^+ \cup \mathbf{R}_{\sqsupset V_1}^-)}$, $p^+(V_1)$, $p^-(V_1)$

5.1.1 Positive Polarity

In the compact conversion, the child of a positive polarity conjunction is never renamed as it can never reduce the number of clauses generated by the conjunction. To show that the Boy de la Tour conversion similarly never renames the children of positive conjunction vertices we look at the conditions for renaming vertex X in the RBC in Figure 10a (the argument for Y is the same; similarly, BX is unsigned but the argument adapts easily to the signed case).

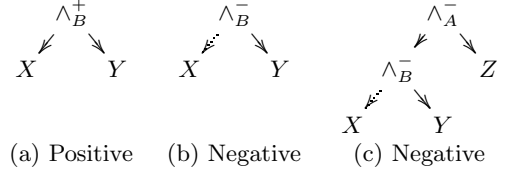


Figure 10. RBC fragments for the optimality proof

We evaluate the benefit $B(T, X)$ in the context $\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)$ substituting the identity from the structure of the graph, $a_X^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} = a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)}$, to find the benefit

$$a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} p^+(X) - (a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} + p^+(X))$$

We start by reducing the condition $B(T, X) > 0$ to the weakened conditions $a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} \geq 2$, $p^+(X) \geq 2$. From the definition of the coefficients we can see that to obtain the former, vertex B must not be renamed. That is $B(T, B) \leq 0$; since $B \notin \mathbf{R}^+ \cup \mathbf{R}^-$, we know that $\mathbf{R}_{\sqsupset B}^+ = \mathbf{R}_{\sqsupset X}^+$ and $\mathbf{R}_{\sqsubseteq B}^- = \mathbf{R}_{\sqsubseteq X}^-$, and hence the non-renaming condition for B is

$$a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} p^+(B) - (a_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} + p^+(B)) \leq 0$$

which together with the conditions above reduces to $p^+(B) \leq 2$. Since B is a conjunction it produces $p^+(X) + p^+(Y)$ clauses; the weakened condition on $p^+(X)$ is thus in conflict with the condition that B is not renamed. We conclude that the Boy de la Tour conversion does not rename the children of positive polarity conjunctions.

5.1.2 Negative Polarity

The compact conversion does not make use of the coefficients a_V^T and b_V^T . We show that the use that the Boy de la Tour conversion makes of these coefficients is limited. Using Figure 10b and the identity from the structure of the graph, $b_X^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} = b_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} p^-(Y)$, we obtain the following expression for $B(T, X)$:

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

This breaks down into two cases. If $b_B^{\text{ren}(T, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} = 1$ then the renaming decision is localised: it is based only on $p^+(X)$ and $p^-(Y)$; this is similar to the condition used for the compact conversion except for the ordering of renamings:

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

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

$$b_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} p^-(B) \leq b_B^{\text{ren}(\psi, \mathbf{R}_{\sqsupset X}^+ \cup \mathbf{R}_{\sqsubseteq X}^-)} + p^-(B)$$

	$p_r^+(E, \mathbf{R}^+, \mathbf{R}^-)$	$p_r^-(E, \mathbf{R}^+, \mathbf{R}^-)$
$sign(E) = +$	$p_r^+(target(E), \mathbf{R}^+, \mathbf{R}^-)$	$p_r^-(target(E), \mathbf{R}^+, \mathbf{R}^-)$
$sign(E) = -$	$p_r^-(target(E), \mathbf{R}^+, \mathbf{R}^-)$	$p_r^+(target(E), \mathbf{R}^+, \mathbf{R}^-)$
	$p_r^+(V, \mathbf{R}^+, \mathbf{R}^-)$	$p_r^-(V, \mathbf{R}^+, \mathbf{R}^-)$
$V \in \mathbf{V}_L$	1	1
$V \in \mathbf{R}^+$	1	—
$V \in \mathbf{R}^-$	—	1
$op(V) = \wedge$	$p_r^+(left(V), \mathbf{R}^+, \mathbf{R}^-) + p_r^+(right(V), \mathbf{R}^+, \mathbf{R}^-)$	$p_r^-(left(V), \mathbf{R}^+, \mathbf{R}^-)p_r^-(right(V), \mathbf{R}^+, \mathbf{R}^-)$
$op(V) = \leftrightarrow$	$p_r^+(left(V), \mathbf{R}^+, \mathbf{R}^-)p_r^-(right(V), \mathbf{R}^+, \mathbf{R}^-) + p_r^-(left(V), \mathbf{R}^+, \mathbf{R}^-)p_r^+(right(V), \mathbf{R}^+, \mathbf{R}^-)$	$p_r^+(left(V), \mathbf{R}^+, \mathbf{R}^-)p_r^+(right(V), \mathbf{R}^+, \mathbf{R}^-) + p_r^-(left(V), \mathbf{R}^+, \mathbf{R}^-)p_r^-(right(V), \mathbf{R}^+, \mathbf{R}^-)$

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

$$\text{COMP}^+(T, E) = \text{COMP}^+(T, target(V))$$

$$\text{COMP}^-(T, E) = \text{COMP}^-(T, target(V))$$

$$\text{COMP}^+(T, V) = \begin{cases} \emptyset & \text{if } V \in \mathbf{V}_L, \text{ or} \\ \text{COMP}^+(T, left(V)) \cup \text{COMP}^+(T, right(V)) & \text{if } r_T^+(V) = 0, \text{ or} \\ \text{COMP}^+(T, left(V)) \cup \text{COMP}^+(T, right(V)) & \text{if } op(V) = \wedge, \text{ or} \\ \text{dis}^{+-}(V) \cup \text{dis}^{-+}(V) \cup \text{COMP}^+(T, left(V)) \cup \text{COMP}^+(T, right(V)) & \text{if } op(V) = \leftrightarrow \end{cases}$$

$$\text{COMP}^-(T, V) = \begin{cases} \emptyset & \text{if } V \in \mathbf{V}_L, \text{ or} \\ \text{COMP}^-(T, left(V)) \cup \text{COMP}^-(T, right(V)) & \text{if } r_T^-(V) = 0, \text{ or} \\ \text{dis}^{--}(V) \cup \text{COMP}^-(T, left(V)) \cup \text{COMP}^-(T, right(V)) & \text{if } op(V) = \wedge, \text{ or} \\ \text{dis}^{++}(V) \cup \text{dis}^{--}(V) \cup \text{COMP}^-(T, left(V)) \cup \text{COMP}^-(T, right(V)) & \text{if } op(V) = \leftrightarrow \end{cases}$$

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

Figure 9. Renaming sets construction for the compact conversion

This holds only when $p^-(B) \leq 2$. Given $p^-(B) = p^+(X)p^-(Y)$ from the structure of the graph we can consider the possibility of renaming under each of the three possible assignments to $p^+(X)$ and $p^-(Y)$; in each case, a contradiction is found either with the renaming condition for X or with the renaming condition for B . Each assignment also fails to satisfy the reduced renaming condition $B'(T, X)$. Thus we *do not need to know the value of b_X^T* in order to make a correct decision about the renaming.

Using this reduced condition, the Boy de la Tour conversion has no restriction on the order of evaluation. It still differs from the compact in that it requires p^+ and p^- rather than p_r^+ and p_r^- . However, we know that renaming only reduces the number of clauses generated by a subgraph: $p^+(V) \geq p_r^+(V, \mathbf{R}_{\square V}^+, \mathbf{R}_{\square V}^-)$. Thus we need consider only the case where $p^+(V)$ is large enough to cause renaming (for the reduced Boy de la Tour condition) but $p_r^+(V, \mathbf{R}_{\square V}^+, \mathbf{R}_{\square V}^-)$ is not (for the compact condition). In all other cases, the two conversions have precisely the same behaviour. Using the situation shown in Figure 10c we can enumerate the possible cases based on the condition $X \in \mathbf{R}^+, A, B, Y, Z \notin \mathbf{R}^-$:

$p^-(Z)$	$p^-(B)$	$p_r^+(B, \mathbf{R}_{\square B}^+, \mathbf{R}_{\square B}^-)$	$p^+(X)$	$p^-(Y)$
2	≥ 3	≤ 2	≥ 3	2
≥ 3	≥ 2	1	Not possible	

For both conversions, the number of clauses generated for A is the same: in Boy de la Tour,

$$p^-(A) = p^+(X) + p^-(Y) + p^-(Z) = p^+(X) + 4$$

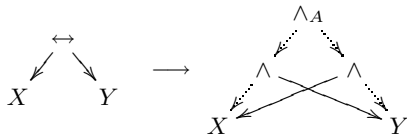
while in the compact conversion

$$p_r^-(A, \mathbf{R}_{\square A}^+, \mathbf{R}_{\square A}^-) = p^+(X) + p^-(Y)p^-(Z) = p^+(X) + 4$$

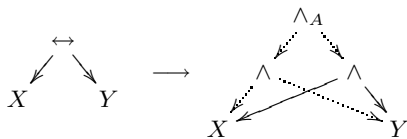
The extra renaming in the Boy de la Tour case is thus redundant, and the compact conversion is optimal in the number of clauses for RBC trees without equivalences.

5.2 Optimality with Equivalences

It is easy to see that the behaviour of compact in the presence of equivalences can be modelled by replacing the equivalence by subgraphs consisting only of conjunctions; for positive polarity:

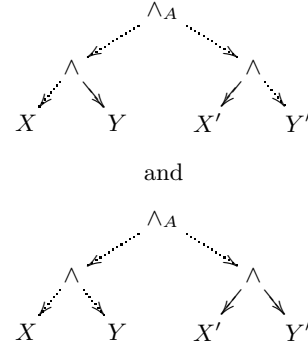


and for negative polarity, inverting the sign of the incoming edge to ensure positive polarity for the result of the substitution:



Note that according to Definition 1, the edges connecting the equivalence vertex to vertices X and Y are unsigned, so

these replacements can be made unconditionally. Following these replacements, the subgraphs X and Y still have the same (zero) polarity that they had before. These transformations violate our sharing restriction, so we begin our analysis by duplicating vertices X and Y to produce the graphs



Since these are now trees of conjunctions, the number of clauses produced is optimal for this arrangement. We need to know whether the minimum number of clauses produced is increased by doing these transformations: if not, the compact conversion must be optimal in the presence of equivalences. This arrangement gives us four subgraphs on which to make (independent) renaming decisions: X, Y, X', Y' . If we examine equivalences directly (as in the Boy de la Tour conversion) the renaming decision is made for X and X' together, and for Y and Y' together. Consider X and X' . Taken together, we rename them if

$$B(X, \text{ren}(T, \mathbf{R}^+, \mathbf{R}^-)) + B(X', \text{ren}(T, \mathbf{R}^+, \mathbf{R}^-)) > 0$$

which has three types of solution:

$$\begin{aligned} B(X, \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) > 0, B(X', \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) \leq 0 \\ B(X, \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) \leq 0, B(X', \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) > 0 \\ B(X, \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) > 0, B(X', \text{ren}(T, \mathbf{R}^+ \cup \mathbf{R}^-)) > 0 \end{aligned}$$

For the first two cases, we note that the benefit of one renaming is no less than the benefit of both renamings; there is no loss (and potentially some gain) in renaming only one polarity. For the third case, we observe that the decisions made by the compact conversion are indeed independent: the bottom-up nature ensures that both decisions can be made before either renaming is performed, and hence the decision taken for the expanded tree is the same as for the original equivalence.

The discussion above assumes that the equivalence under consideration has non-zero polarity. That is, we can strengthen our optimality result from RBCs without equivalences to RBCs without *nested equivalences*. This follows from the linear size transformation for equivalences given above which reduces the problem to an RBC without equivalences, together with the result that no greater benefit is made available by not doing this transformation.

Of course, to be able to claim optimality in general we must also consider the case of zero polarity (hence nested) equivalences. However, it is clear from the definitions that the children of such a vertex would have $r_T^+ = 2$ and $r_T^- = 2$, even for an RBC without sharing. We will see in the following section that this situation causes the renaming of nontrivial child vertices both positively and negatively, which is optimal.

5.3 RBCs with Sharing

The renaming sets constructions given above assume a tree-structured RBC. To handle general RBCs, we need to find the optimal conversion from a graph to a tree. The basic approach for such a conversion is the repetition of multiply-referenced subgraphs with the alternative of renaming the subgraph in the same way as discussed above. The clause counting functions in Table 1 make no assumptions about sharing, so is equivalent to the repetition solution. The benefit of renaming a vertex V in an RBC rooted at T is thus

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

As before, we can refine this by introducing coefficients, but the definitions depend on the number of incoming edges to a vertex. In fact, the coefficients are equivalent to those in Table 2 summed over the incoming edges. Rather than considering these functions directly, we approximate them by using the reference counting functions. a_V^T and b_V^T are now the average number of repetitions of the clauses of V . Thus

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

with renaming reducing the reference count of V to 1 in appropriate polarities. We can consider the positive and negative polarities independently as we are extending the compact conversion.

Positive polarity benefit: $a_V^T r_T^+ p^+(V) - (a_V^T r_T^+ + p^+(V))$

Negative polarity benefit: $b_V^T r_T^- p^-(V) - (b_V^T r_T^- + p^-(V))$

If $r_T^s = 1$ the fragment is a tree and the existing analysis stands. Otherwise, we have the following cases:

r_T^+	a_V^T	$B(T, V) > 0$	r_T^-	b_V^T	$B(T, V) > 0$
2	1	$p^+(V) \geq 3$	2	1	$p^-(V) \geq 3$
2	> 1	$p^+(V) \geq 2$	2	> 1	$p^-(V) \geq 2$
≥ 3	≥ 1	$p^+(V) \geq 2$	≥ 3	≥ 1	$p^-(V) \geq 2$

That is, vertices with multiple inedges of the same polarity should be renamed if they generate more than one clause in that polarity; an exception is made if there are exactly two inedges with each referencing the vertex once. In this case renaming is not performed for size two. This special case is problematic — we have until now been able to remove all use of the coefficients a_V^T and b_V^T . In fact, if we relax our requirement that renaming always reduces the number of clauses, this special case can be eliminated: renaming does not, in this case, increase the number of clauses generated. The resulting condition $r_T^s(V) \geq 2$, $p^s(V) \geq 2$ $s \in \{+, -\}$ is, as with the refinement of the Boy de la Tour conversion given above, agnostic with regard to the order in which vertices are analysed.

Combining the multiple-reference renaming with the compact conversion requires some care to ensure that optimality is maintained. We must consider those vertices which are renamed during conversion to a tree that, had they not been renamed, would have resulted in the production of fewer clauses. This occurs only when $p_r^s(V, \text{COMP}^+(T, V), \text{COMP}^-(T, V)) = 1$ but $p^s(V) \geq 2$. The reverse situation (a renaming that should have been performed during tree construction was omitted) does not occur

since $p_r^s(V, \text{COMP}^+(T, V), \text{COMP}^-(T, V)) \leq p^s(V)$. The optimal number of clauses is thus generated if the condition $p_r^s(V, \text{COMP}^+(T, V), \text{COMP}^-(T, V)) \geq 2$ is used in place of $p^s(V) \geq 2$. For this to be possible, the conversion to a tree must be performed bottom-up — the graph below V must already be a tree in order to compute the conversion condition required for converting V itself. An efficient implementation, running the two algorithms simultaneously, remains in $O(|V|)$.

6 Evaluation and Conclusions

Table 4 lists the result of applying the conversion algorithms in NuSMV with the SAT solver zChaff [14]. The problems are the standard DME benchmark and an industrial problem from the *Texas-97* benchmark suite [2] (MSI), and two deadlock problems from [12] (Elevator and Mmgmt). Where the SNF [9] encoding is used, this is indicated and we can see how the compact conversion helps to narrow the gap between this and the standard BMC encoding. Unsurprisingly, the compact conversion consistently generates fewer clauses and the solving times are also better in most cases, sometimes dramatically so. For the deadlock problems, however, the larger conversions perform significantly better. These problems are solvable, and it is likely to be a coincidence of variable ordering heuristics. Nevertheless it points a direction for further research into CNF conversions.

Despite optimising a problem attribute that is not directly connected to the solving time — the number of clauses — the optimal algorithm produces a set of clauses that are in most cases more quickly solved by zChaff. 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.

REFERENCES

- [1] Parosh Aziz Abdulla, Per Bjesse, and Niklas Eén, ‘Symbolic reachability analysis based on SAT-solvers’, in *Tools and Algorithms for the Construction and Analysis of Systems, 6th International Conference, TACAS’00*, eds., S. Graf and M. Schwartzbach, volume 1785 of *Lecture Notes in Computer Science*, pp. 411–425. Springer-Verlag Inc., (March 2000).
- [2] Adnan Aziz et al. Examples of HW verification using VIS, 1997. <http://vlsi.colorado.edu/~vis/texas-97/>
- [3] 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).
- [4] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu, ‘Symbolic model checking without BDDs’, in *Tools and Algorithms for the Construction and Analysis of Systems. 5th International Conference, TACAS’99*, ed., W.R. Cleaveland, volume 1579 of *Lecture Notes in Computer Science*, pp. 193–207. Springer-Verlag Inc., (July 1999).
- [5] Thierry Boy de la Tour, ‘An optimality result for clause form translation’, *Journal of Symbolic Computation*, **14**, 283–301, (1992).
- [6] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Roberto Sebastiani, ‘Improving the encoding of LTL model checking into SAT’, in *Third International Workshop on Verification, Model Checking and Abstract Interpretation*, ed., Agostino Cortesi, volume 2294 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., (January 2002).

Table 4. Benchmark results for three clause form conversions

Name	k	SAT	Definitional		SP		Compact	
			Clauses	Time (s)	Clauses	Time (s)	Clauses	Time (s)
DME (Priority 1, SNF)	13	Y	18 961	0.05	7 257	0.01	6 005	0.01
DME (Priority 1)	13	Y	22 043	0.05	8 189	0.01	6 630	0.01
DME (Priority)	52	Y	234 515	4.03	79 507	5.56	52 313	0.77
DME (Priority, SNF)	52	Y	75 121	2.63	28 785	1.32	23 789	0.47
DME (Access)	40	N	70 808	16.34	25 149	5.11	21 268	1.72
DME (Access, SNF)	40	N	58 884	14.14	22 484	2.63	18 640	1.13
MSI (Request A)	20	N	1 423 852	53.97	487 910	13.43	438 045	11.25
MSI (Request A, SNF)	20	N	1 422 861	80.28	487 915	20.49	438 066	13.64
MSI (Request B)	20	Y	1 423 187	174.2	488 011	40.23	438 169	49.80
MSI (Request B, SNF)	20	Y	1 424 359	174.7	488 288	29.37	438 423	23.11
Elevator 3	14	Y	230 104	1424.8	84 441	129.8	83 035	262.9
Elevator 4	17	Y	658 181	1800	239 058	0.48	237 025	826.3
Mmgt 3	10	Y	44 556	28.2	16 770	794.4	16 116	376.9
Mmgt 4	12	Y	70 735	878.3	26 655	483.1	25 652	734.1

- [7] Martin Davis, George Logemann, and Donald Loveland, ‘A machine program for theorem-proving’, *Communications of the ACM*, **5**, 394–397, (1962).
- [8] L. Drake, A. M. Frisch, and T. Walsh, ‘Adding resolution to the DPLL procedure for boolean satisfiability’, in *Proceedings of the Fifth International Conference on the Theory and Applications of Satisfiability Testing (SAT 02)*, Lecture Notes in Computer Science, (2002).
- [9] Alan Frisch, Daniel Sheridan, and Toby Walsh, ‘A fixpoint based encoding for bounded model checking’, in *Formal Methods in Computer-Aided Design; 4th International Conference, FMCAD 2002*, eds., M D Aagaard and J W O’Leary, volume 2517 of *Lecture Notes in Computer Science*, pp. 238–254, Portland, OR, USA, (November 2002). Springer.
- [10] I.P. Gent, ‘Arc consistency in sat’, Technical Report APES-39A-2002, APES Research Group, (June 2002). Available from <http://www.dcs.st-and.ac.uk/apes/apesreports.html>.
- [11] T. A. Juntilla. Boolean circuit tools (including bczchaff). <http://www.tcs.hut.fi/~tjunttil/circuits>, May 2003.
- [12] Toni Jussila, Keijo Heljanko, and Ilkka Niemelä, ‘BMC via on-the-fly determinization’, in *First International Workshop on Bounded Model Checking*, (July 2003).
- [13] Henry Kautz and Bart Selman, ‘Planning as satisfiability’, in *Proceedings of the Tenth European Conference on Artificial Intelligence*, ed., J. Lloyd, pp. 359–379, (1992).
- [14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, ‘Chaff: Engineering an efficient SAT solver’, in *39th Design Automation Conference*, Las Vegas, (June 2001).
- [15] Andreas Nonnengart, Georg Rock, and Christoph Weidenbach, ‘On generating small clause normal forms’, in *Fifteenth International Conference on Automated Deduction*, eds., Claude Kirchner and Hélène Kirchner, volume 1421 of *Lecture Notes in Artificial Intelligence*, pp. 397–411. Springer-Verlag, (1998).
- [16] David A. Plaisted and Steven Greenbaum, ‘A structure-preserving clause form translation’, *Journal of Symbolic Computation*, **2**(3), 293–304, (September 1986).
- [17] Daniel Sheridan and Toby Walsh, ‘Clause forms generated by bounded model checking’, in *Eighth Workshop on Automated Reasoning*, ed., Andrei Voronkov, (2001).
- [18] Ofer Shtrichman, ‘Tuning SAT checkers for bounded model checking’, in *Proceedings of the 12th International Conference on Computer Aided Verification (CAV’00)*, volume 1855 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., (2000).
- [19] SPASS: An automated theorem prover for first-order logic with equality. <http://spass.mpi-sb.mpg.de/>, 1998.
- [20] M. N. Velev, ‘Efficient translation of Boolean formulas to CNF in formal verification of microprocessors’, in *Asia and South Pacific Design Automation Conference (ASP-DAC ’04)*, (January 2004).