MULTIRESOLUTION FOR SAT CHECKING

PHILIPPE CHATALIC and LAURENT SIMON
Laboratoire de Recherche en Informatique
U.M.R. CNRS 8623, Université Paris-Sud
91405 Orsay Cedex, France
email : {chatalic, simon}@lri.fr

Received (Oct. 4th 2001)
Revised (Nov. 12th 2001)

This paper presents a system based on new operators for handling sets of propositional clauses compactly represented by means of ZBDDs. The high compression power of such data structures allows efficient encodings of structured instances. A specialized operator for the distribution of sets of clauses is introduced and used for performing multiresolution on clause sets. Cut eliminations between sets of clauses of exponential size may then be performed using polynomial size data structures. The ZRes system, a new implementation of the Davis-Putnam procedure of 1960, solves two hard problems for resolution, that are currently out of the scope of the best SAT provers.

Keywords: Resolution, Multiresolution, SAT, ZBDD, Compression, Complexity

1. Introduction

Recent years have seen a lot of work using propositional logic as a framework for knowledge representation and problem solving. The poor expressive power of propositional logic is counterbalanced by its simplicity which allows simple, but efficient, provers to be constructed. In particular, the SAT problem has been the focus of much interest, ranging from refinement and optimization of complete procedures [29] to the introduction of new incomplete methods [21]. Hard random instances have been identified [17] and reference problems have been gathered in benchmarks, subject to program competitions (e.g. [7,28]). If its central place in complexity theory is probably part of the motivation for such work, the resolution of real world problems is also an attractive challenge for SAT.

In the area of complete methods, most current SAT provers are based on the Davis, Logeman and Loveland procedure (DLL) [14]. One reason of success of such solvers is their ability to limit memory usage. This is due to the fact that they essentially make choices, resulting in simplifications and unit propagations. The difficulty comes from the number of possible choices which is exponential. The role of heuristics is then central. However, heuristics which are efficient on random problems are seldom appropriate for structured instances, on which dynamic heuristics based on learning techniques give better results [29]. Other problems, such as prime implicants/implicates generation or knowledge base compilation, often handle large sets of clauses. Then, the difficulty rather comes from the size of such sets that may grow exponentially. Whatever the goal, one is rapidly faced with combinatorial explosion. On structured instances, the only way to deal with such a growth seems to be taking the structure of the available information into account. If heuristics can prune the search space efficiently, savings in space may be achieved by means of *compression* algorithms.

This paper focuses on data structures used for encoding sets of clauses and on associated operators. We are particularly interested in instances with a special structure. Our feeling is that such instances should show some regularities through the encoding. An appropriate data structure should be able to take advantage of such regularities to handle a more compact representation of the encoded formula. For cnf formulas, such regularities may correspond to subsets of literals that appear in many different clauses. We propose to use a data structure allowing the factorization of common subsets of literals. *Tries* structures [5] enable the factorization of clauses beginning with the same sequence of literals. They remain the state-of-the-art data structures for subsumption checking [29]. We further generalize this idea to factorize simultaneously ends of clauses. In practice, we use a variant of Binary Decision Diagrams (BDDs) [1], called ZBDDs [16], to store sets of clauses by means of their characteristic function. We show that large sets of clauses corresponding to structured instances may be efficiently compressed in that way. Moreover, such structures are also well suited to subsumption checking, and all set operations can be realized as operations on ZBDDs. We introduce a new operator for performing multiple resolutions on sets of clauses represented by ZBDDs, in a single step. We use this operator in an implementation of the original Davis and Putnam algorithm [4] (DP as opposed to DLL).

The next section presents the principles of multiresolution, an inference rule on sets of clauses, that is used in our implementation of DP. In the third section, basic principles of BDDs and ZBDDs structures are first recalled. A technique to encode sets of clauses by means of ZBDDs, as well as new operators are introduced. In the fourth section we show that expressing cut-elimination using these operators makes it possible to perform multiresolution on sets of clauses. We illustrate its use in a new implementation of DP, called ZRes. The last sections are devoted to empirical studies. Two classes of hard instances for resolution, namely the Pigeon Hole problem and the Urquhart problem, are tested and compression capabilities of ZBDDs are evaluated.

2. Multiresolution

The resolution inference rule plays a central role in many works based on clausal representations of boolean formulas. It may be characterized as the rule $(x \vee c_1) \wedge (\neg x \vee c_2) \vdash c_1 \vee c_2$. Its power comes from the fact that the inferred formula $c_1 \vee c_2$ is still a clause. The way this rule is defined however implicitely suggests that it has to be used on pairs of clauses containing complementary literals. Many practical results, as well are theoretical results, are based on this assumption.

This work is based on an alternative inference rule called **multiresolution** which aim is basically to perform resolution on sets of clauses, instead of just pairs of clauses. Resolution may be seen as specialisation of the the more general *cut* inference rule $(x \vee f) \wedge (\neg x \vee g) \vdash_{cut} f \vee g$ which holds for any boolean formulas f and g. Multiresolution may be considered as a variant of another specialisation of the cut rule, to the case where f and g correspond to sets of clauses (assimilated to conjunctions of clauses). Note that in that case, the inferred formula $f \vee g$ is no more a conjunction of clauses, but a disjunction of conjunctions of clauses. It is however not difficult to transform this formula into an equivalent conjunction of clauses, by using the distributivity property of the $\vee$ operator over the $\wedge$ operator.

Definition 1 (clause distribution $\otimes$) Let Σ_1 and Σ_2 be two sets of clauses, the **clause distribution** of Σ_1 by Σ_2 is the set of clauses obtained as the union of the literals of some clause of Σ_1 with those of some clause of Σ_2, and that are not tautologies. It is denoted by $\Sigma_1 \otimes \Sigma_2$.

Property 1 Let Σ_1 and Σ_2 be two sets of clauses. Then the formula $\Sigma_1 \vee \Sigma_2$ and $\Sigma_1 \otimes \Sigma_2$ are equivalent formulas.

This proposition follows from the distributitivity properties of the logical connectives and the fact that removing tautologies does not affect the meaning of a set of formulas. This may be viewed as a systematic way to transform a disjunction of sets of clauses into an equivalent set of clauses.

Definition 2 (Multiresolution) Let Σ_1 and Σ_2 be two sets of clauses. The multiresolution inference rule is defined by: $(x \vee \Sigma_1) \wedge (\neg x \vee \Sigma_2) \vdash_{\mathcal{MR}} \Sigma_1 \otimes \Sigma_2$.

Multiresolution is thus an inference rules that applies to sets of clauses containing complementary literals and infers a set of clauses. It is of course closely related to classical resolution since any deduction by multiresolution is equivalent to a set of deductions by classical resolution and reciprocally, any classical resolution of a non tautological clause may be performed by multiresolution on singleton sets of clauses.

From an implementation point of view, multiresolution can of course be achieved by means of classical resolution. But if the clause distribution operator may be implemeted in an efficient manner, then it is possible to implement multiresolution directly at the set level, without having to consider clauses one by one.

3. Representing sets of clauses with BDDs

A Binary Decision Diagrams [1,20] is a directed acyclic graph with labeled nodes, a unique source node, and such that each node is either a sink node or an internal node (denoted by $\Delta(x, n_1, n_2)$) having two children (n_1, n_2) and a *label* x. The two children are respectively connected to their parent node through a 1-arc and a 0-arc. Intuitively, each label corresponds to some decision function and n_1, n_2 caracterize the cases to be considered depending on the value of this function.

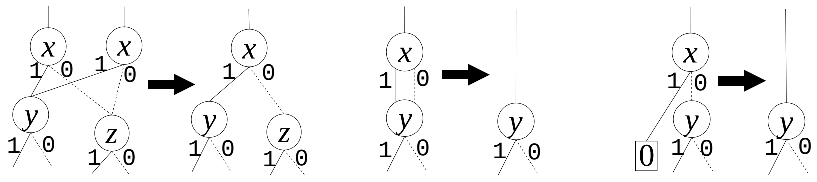
3.1. *Encoding boolean functions with BDDs*

BDDs encoding boolean functions have only two sinks, labeled by 1 and 0, and their internal nodes are labeled with boolean variables $\{x, y, \ldots\}$. The classical semantics interprets sinks 1 and 0 respectively as *true* and *false*, and any internal node $n = \Delta(x, n_1, n_2)$ as the function $f = \textit{if } x \textit{ then } f_1 \textit{ else } f_2$, where f_1 and f_2 are the respective interpretations of n_1 and n_2. As a consequence, each path from the source node to the sink 1 of a BDD corresponds to a model of the encoded formula. Many refinements of BDDs have been proposed in the literature. For instance, given an ordering on variables, Ordered BDDs (OBDDs) require the label of any node to be smaller than the labels of its children. OBDDs may thus be viewed as binary trees encoding the Shannon decomposition [20] of the initial formula.

To obtain more compact representations, additional reduction rules may be used. Reduced OBDDs (ROBDDs) require the graph not to contain any isomorphic subgraphs (*node sharing* rule, fig 1-a) or any useless node of the form $\Delta(x, n, n)$, that do not care about its label value (*node elimination* rule, fig 1-b). Zero-supressed BDDs (ZBDDs) [16] replace he node elimination rule by the *Z-elimination* rule (fig. 1-c), for which useless nodes are those of the form $\Delta(x, 0, n)$. This means that, variables that do not appear explicitly on a path from the source node to a sink node are by default interpreted as *false* on this path. In the following, without further precisions, we simply use the term BDD in place of ROBDD.

The main advantage of using a BDD for encoding a propositional formula is that it can describe very large sets of models in a very compact way. However this requires computing the Shannon normal form of the formula, which may be very expensive if the formula is in conjunctive normal form (cnf). From the SAT point of view, since the BDD encodes the set of all models, this is as difficult as counting them, which is #P-complete.

3.2. *Encoding sets of clauses with ZBDDs*



a.Node sharing *b*.Node elimination *c*.Z-elimination

Figure 1: Reductions rules in BDDs

To benefit from the high compression capabilities of BDDs while avoiding the computation of Shannon normal forms, we propose an alternative encoding and use ZBDDs to encode sets of clauses corresponding to cnf formulas. We still use two sink nodes 1 and 0, but internal nodes are labeled with literals (instead of variables). Intuitively, each path, from the source node to the 1 sink, represents a clause containing all the literals labeling the parent nodes of 1-arcs on this path. One may notice that in [3,20], ZBDDs are also used as data structures for encoding sequences of literals corresponding to sets of prime implicants. However all these approaches first compute the ROBDD of the initial formula, and use a property of the Shannon decomposition (the *decomposition theorem*) to derive the set of prime implicants. Our approach is different since the ZBDD is directly constructed from the original cnf.

Given an initial ordering $x_1 < \ldots < x_n$ on the set of variables, we consider the extended literal ordering corresponding to $x_1 < \neg x_1 < \ldots < x_n < \neg x_n$. In the following, without further notice, we assume that we only consider ZBDDs labelled by literals ordered in this way.

Definition 3 (Support) Given a ZBDD Δ, the **support** of Δ (denoted by $sup(\Delta)$ is the set of its labels. If the support of Δ is not empty we denote by $label(\Delta)$ the smallest element of $sup(\Delta)$.

Definition 4 (Size) Given a ZBDD Δ, the **size** of Δ is the number $nn(\Delta)$ of its internal nodes.

The semantics we use interprets such ZBDDs as sets of clauses and may be formalized by:

Definition 5 (Semantics)

- $[\![0]\!] = \emptyset$ (the empty set of clauses)
- $[\![1]\!] = \{\Box\}$ (the set reduced to the empty clause)
- $[\![\Delta(l, A, B)]\!] = \{l \vee [\![A]\!]\} \cup [\![B]\!]$,

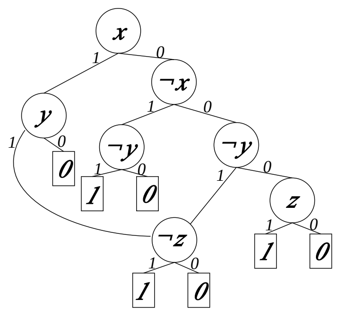


Figure 2: A ZBDD encoding of S_1

where $\{l \vee [\![A]\!]\}$ denotes the set of clauses obtained by adding the literal l to each clause of $[\![A]\!]$.

Encoding a set of clauses via a ZBDD essentially gives the possibility to factorize common beginnings and ends of clauses. For instance, with the initial variable ordering $x < y < z$, the set $S_1 = \{x \vee y \vee \neg z, \neg x \vee \neg y, \neg y \vee \neg z, z\}$ may be represented[1] by the ZBDD of figure 2.

Let us notice that the meaning of a ZBDD $[\![\Delta(l, A, B)]\!]$ corresponds to the union of two disjoint sets of clauses, since $\{l \vee [\![A]\!]\}$ corresponds to clauses that all contain the literal l, while on the contrary, none of the clauses of $[\![B]\!]$ contains the literal l. An internal node thus corresponds to the decision function partitioning the whole set of clauses encoded by the ZBDD corresponding to this node into the subset of clauses containing the label of this node and the subset of clauses not containing this label.

This property may be used practically during the encoding of a set of clauses Σ as a ZBDD. The first step is to determine the lowest literal symbol l appearing in Σ (for the considered literal order). The set Σ is then partitionned into Σ_l and $\overline{\Sigma_l}$, where Σ_l denotes the subset of clauses containing the literal l and $\overline{\Sigma_l} = \Sigma \setminus \Sigma_l$. Now let us denote by $\Sigma_l^{'}$ the set obained by suppressing le literal l from any clause of Σ_l. Then we have $\Sigma = \{l \vee \Sigma_l^{'}\} \cup \overline{\Sigma_l}$. We may construct recursively the two ZBDD Δ_1 and Δ_2 encoding respectively $\Sigma_l^{'}$ and $\overline{\Sigma_l}$. The final ZBDD encoding Σ is then obtained by creating a new node labelled by l, the sons of which correspond to Δ_1 and Δ_2, after application of the possible reduction rules, in particular the node sharing rule. The encoding of a set of clauses as a ZBDD is thus a recursive process. There are two terminal cases. If Σ is empty it is encoded by the 0 sink. If Σ_l contains the (single literal) clause l, then Σ_l is encoded by the 1 sink.

Property 2 Let Δ be the ZBDD constructed from a label l and two ZBDD A and B, then $nn(\Delta) \leq 1 + nn(A) + nn(B)$.

[1]Sink nodes 1 and 0 have been duplicated for a better readability.

By construction, if no reduction rule applies, Δ contains all nodes of A, all nodes of B and one additional node for the source node. Then we have $nn(\Delta) = 1 + nn(A) + nn(B)$. However, if any reduction rule applies some of the nodes may be eliminated and then $nn(\Delta)$ may be smaller.

Let us now consider the two functions :

Definition 6 The function nc is defined by:

- $nc(0) = 0$
- $nc(1) = 1$
- $nc(\Delta(l, A, B)) = nc(A) + nc(B)$

Given the above semantics, it can be easily shown that $nc(\Delta)$ corresponds to the *number of clauses* of $[\![\Delta]\!]$. This naturally follows from the fact that $\{l \vee [\![A]\!]\}$ and $[\![B]\!]$ are disjoint sets.

Property 3 If $\Delta = \Delta(l, A, B)$ then $nc(A) > 0$. If $\Delta \neq 0$ then $nc(\Delta) > 0$.

Proof. This is a direct consequence of the Z-elimination rule, since it is impossible to have an internal node of the form $\Delta(l, 0, B)$. $\square$

Definition 7 The function nl is defined by:

- $nl(0) = 0$
- $nl(1) = 0$
- $nl(\Delta(l, A, B)) = nc(A) + nl(A) + nl(B)$

It can be easily shown that $nl(\Delta)$ corresponds to *number of literals* of $[\![\Delta]\!]$. Again, this follows from the fact that $\{l \vee [\![A]\!]\}$ and $[\![B]\!]$ are disjoint sets and that the literal l appears exactly in $nc(A)$ clauses.

Property 4 If $\Delta = \Delta(l, A, B)$ then $nl(A) < nl(\Delta)$ and $nl(B) < nl(\Delta)$

Proof. This is a direct consequence of the property 3. $\square$

A more interesting property is the following, which states that using ZBDD may be an adequate way to obtain a *compressed representation* of a set of clauses:

Property 5 The size of a ZBDD Δ is never bigger than the size of the set of clauses that it encodes, i.e. $nn(\Delta) \leq nl(\Delta)$.

Proof. This may be proved by induction on the size of the support $n = |sup(\Delta)|$. It clearly holds for $n = 0$. Let us suppose that the property holds for any ZBDD such that $|sup(\Delta)| < n$ and let $\Delta(l, A, B)$ be a ZBDD such that $|sup(\Delta(l, A, B))| = n$. By definition $nl(\Delta(l, A, B)) = nc(A) + nl(A) + nl(B)$. By property 3 $nc(A) \geq 0$.

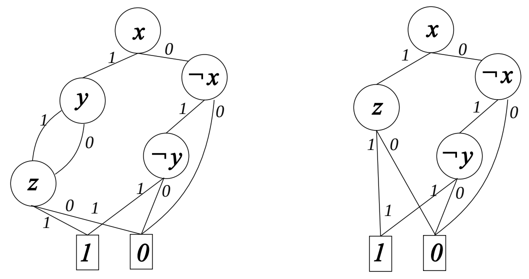


Figure 3: Σ_2 encoding before (a) and after (b) node elimination

Since the size of the support of A and B is smaller than n, by induction hypothesis, $nn(A) \leq nl(A)$ and that $nn(B) \leq nl(B)$. Thus $1+nn(A)+nn(B) \leq nl(\Delta(l, A, B))$. Hence, by property 2, $nn(\Delta(l, A, B)) \leq nl(\Delta(l, A, B))$. $\square$

Interpreting ZBDDs as sets of clauses has a significant impact on the encoding, one of which is the easy detection of some kinds of subsumed clauses. Let us consider the set $\Sigma_2 = \{\neg x \vee \neg y, x \vee y \vee z, x \vee z, \neg x \vee \neg y \vee z\}$. The corresponding ZBDD is represented on figure 3-a. Note that the clause $\neg x \vee \neg y \vee z$, which is subsumed by the clause $\neg x \vee \neg y$, cannot be explicitely represented. More generally, any clause c_1 of length $n+k$, such that the n first literals correspond to another clause c_2 cannot be explicitly represented. But, since c_1 is subsumed by c_2, it can be simply removed. Another simplification may also be performed on the node labelled by y. The path going through its 1-arc corresponds to the clause $x \vee y \vee z$ while the path going through its 0-arc corresponds to the clause $x \vee z$, which clearly subsumes the previous one. The ZBDD obtained after elimination of the y node (fig. 3-b) thus corresponds to an equivalent set of clauses. More generally, any node of the form $\Delta(x, A, A)$ corresponds to the set of clauses $\{x \vee [\![A]\!]\} \cup [\![A]\!]$. So, each clause of the first set is subsumed by a clause of the second set. With our semantics, it is thus possible to take advantage of both the Z-elimination rule and the ROBDD node-elimination rule.

Another easy simplification is the elimination of tautologies. A ZBDD of the form $\Delta(x, \Delta(\neg x, A, B), C)$ corresponds to the union of three sets of clauses : $\{x \vee \{\neg x \vee [\![A]\!]\}\} \cup \{x \vee [\![B]\!]\} \cup [\![C]\!]$. However, clauses containing both x and $\neg x$ are tautologies. A simple way to eliminate them is to apply the *tautology elimination rule*, which systematically replaces nodes of the form $\Delta(x, \Delta(\neg x, A, B), C)$ by $\Delta(x, B, C)$. In the following, we assume that all considered ZBDDs are constructed by systematically applying the appropriate reduction rules.

3.3. *Operations on clause sets*

Minato has shown in [16] that basic set operations (union, intersection,...) may be realized as operations on ZBDDs. We here introduce new recursive operators that take advantage of our special semantics, but that could not be used with usual semantics of ZBDDs. In the following, literals are denoted by $\{l, m, \ldots\}$. We also make use of the following lemma :

Lemma 1 Let $A = \Delta(l, A_1, A_2)$ and $B = \Delta(m, B_1, B_2)$ be two ZBDDs such that $l < m$. No clause of $\{l \vee [\![A_1]\!]\}$ subsumes a clause of $[\![B]\!]$.

Proof. This naturally results from the fact that all clauses of $\{l \vee [\![A_1]\!]\}$ contain the literal l while clauses of $[\![B]\!]$ may only contain literals which are greater or equal to m. Since $l < m$, none of them contains the literal l and can be subsumed by a clause containing l. $\square$

As a direct consequence of this lemma we have :

Corollary 1 Let $A = \Delta(l, A_1, A_2)$ be a ZBDD. No clause of $\{l \vee [\![A_1]\!]\}$ subsumes a clause of $[\![A_2]\!]$.

We now introduce a first binary operator on ZBDD, that is used in the definition of the next operators:

Definition 8 ($\setminus\!\!\setminus$ operator) Let A and B be two ZBDD, the ZBDD $A \setminus\!\!\setminus B$ is defined by:

T_1 : $0 \setminus\!\!\setminus A = 0$

T_{2a}: $1 \setminus\!\!\setminus 1 = 0$

T_{2b}: $1 \setminus\!\!\setminus A = 1$,if $A \neq 1$

T_{3a}: $\Delta(l, A_1, A_2) \setminus\!\!\setminus 0 = \Delta(l, A_1, A_2)$

T_{3b}: $\Delta(l, A_1, A_2) \setminus\!\!\setminus 1 = 0$

R_1 : $(l > m)\, \Delta(l, A_1, A_2) \setminus\!\!\setminus \Delta(m, B_1, B_2) = \Delta(l, A_1, A_2) \setminus\!\!\setminus B_2$

R_2 : $(l < m)\, \Delta(l, A_1, A_2) \setminus\!\!\setminus \Delta(m, B_1, B_2) =$
$\Delta(l,\, A_1 \setminus\!\!\setminus \Delta(m, B_1, B_2),\, A_2 \setminus\!\!\setminus \Delta(m, B_1, B_2))$

R_3 : $\Delta(l, A_1, A_2) \setminus\!\!\setminus \Delta(l, B_1, B_2) = \Delta(l,\, (A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2,\, (A_2 \setminus\!\!\setminus B_2))$

Definition 9 (Subsumed difference) Given two sets of clauses Σ_1 and Σ_2, the **subsumed difference** $\Sigma_1 \widehat{\setminus\!\!\setminus} \Sigma_2$ is the set of clauses obtained by removing from Σ_1 all clauses of Σ_2 as well as all clauses subsumed by some clause of Σ_2.

Theorem 1 Let A and B be two ZBDDs, $[\![A \setminus\!\!\setminus B]\!] = [\![A]\!] \widehat{\setminus\!\!\setminus} [\![B]\!]$.

Proof. We first show that the property holds for all pairs of ZBDD corresponding to one of the terminal cases $T_1, T_{2a}, T_{2b}, T_{3a}, T_{3b}$. If $A = 0$, $[\![A]\!] = \emptyset$ and we must have $[\![A]\!] \widehat{\setminus\!\!\setminus} [\![B]\!] = \emptyset$. This is in accordance with the rule T_1. If $A \neq 0$ but $B = 0$,

$[\![B]\!] = \emptyset$ and thus no clause of $[\![A]\!]$ may be subsumed by some clause of $[\![B]\!]$. Thus we have $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = [\![A]\!]$. This is in accordance with the rule T_{2b} and T_{3a}. If $B = 1$, then $[\![B]\!] = \{\Box\}$. Since $\Box$ subsumes any clause, we must have $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = \emptyset$. This is in accordance with the rule T_{2a} and T_{3b}.
The rest of the proof proceeds by induction on $n = |sup(A)| + |sup(B)|$ (i.e. the sum of the size of the support of A and B). The property holds if $n = 0$, which corresponds to one of the above considered terminal cases. Let $n \geq 0$, let us suppose that the property holds for any ZBDD A and B such that $|sup(A)| + |sup(B)| < n$ and let us consider two ZBDD A and B such that $|sup(A)| + |sup(B)| = n$. The case where $|sup(A)| = 0$ corresponds to the terminal case T_1 or T_{2b}. The case where $|sup(B)| = 0$ corresponds to the terminal case T_{3a} or T_{3b}. If both $|sup(A)| > 0$ and $|sup(B)| > 0$, A and B are respectively of the form $\Delta(l, A_1, A_2)$ and $\Delta(m, B_1, B_2)$. Three cases may then be considered.

- If $l > m$, the rule R_1 applies and $[\![A \setminus\!\!\setminus B]\!] = [\![A \setminus\!\!\setminus B_2]\!]$. Now let us consider $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!]$. Since $l > m$, by lemma 1, no clause of $\{m \vee [\![B_1]\!]\}$ may subsume a clause of $[\![A]\!]$ and thus $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = [\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!]$. Since $|sup(B_2)| < |sup(B)|$, we have $|sup(A)| + |sup(B_2)| < n$. Hence, by induction hypothesis, $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!] = [\![A \setminus\!\!\setminus B_2]\!]$.

- If $l < m$, the rule R_2 applies and $[\![A \setminus\!\!\setminus B]\!] = [\![\Delta(l,\, A_1 \setminus\!\!\setminus B,\, A_2 \setminus\!\!\setminus B)]\!]$. But since $l < m$, no clause of $[\![B]\!]$ does contain the literal l. Thus if some clause of $[\![B]\!]$ subsumes some clause of $[\![A]\!]$ containing the literal l, it necessarily subsumes the same clause without the literal l in $[\![A_1]\!]$. Hence, $(\{l \vee [\![A_1]\!]\} \cup [\![A_2]\!]) \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = (\{l \vee ([\![A_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!])\}) \cup ([\![A_2]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!])$. Since $|sup(A_1)| + |sup(B)| < n$ and $|sup(A_2)| + |sup(B)| < n$, by induction hypothesis, $[\![A_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = [\![A_1 \setminus\!\!\setminus B]\!]$ and $[\![A_2]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = [\![A_2 \setminus\!\!\setminus B]\!]$. Hence, $(\{l \vee [\![A_1]\!]\} \cup [\![A_2]\!]) \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = \{l \vee [\![A_1 \setminus\!\!\setminus B]\!]\} \cup [\![A_2 \setminus\!\!\setminus B]\!]$. By definition, this corresponds to $[\![\Delta(l,\, A_1 \setminus\!\!\setminus B,\, A_2 \setminus\!\!\setminus B)]\!]$.

- If $l = m$, the rule R_3 applies, and $[\![A \setminus\!\!\setminus B]\!] = [\![\Delta(l,\, (A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2, A_2 \setminus\!\!\setminus B_2)]\!]$. Now we have $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = (\{l \vee [\![A_1]\!]\} \cup [\![A_2]\!]) \mathbin{\widehat{\setminus\!\!\setminus}} (\{l \vee [\![B_1]\!]\} \cup [\![B_2]\!])$. By lemma 1 no clause of$\{l \vee [\![B_1]\!]\}$ may subsume a clause of $[\![A_2]\!]$. Thus $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = \{(\{l \vee [\![A_1]\!]\} \mathbin{\widehat{\setminus\!\!\setminus}} \{l \vee [\![B_1]\!]\}) \cup [\![A_2]\!]\} \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!]$.
Moreover, if a clause of $\{l \vee [\![B_1]\!]\}$ subsume some clause of $\{l \vee [\![A_1]\!]\}$ the subsumption also occurs on the same pair of clauses without the literal l.
Thus, $\{l \vee [\![A_1]\!]\} \mathbin{\widehat{\setminus\!\!\setminus}} \{l \vee ([\![B_1]\!]\} = \{l \vee ([\![A_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_1]\!])\}$. Since $|sup(A_1)| + |sup(B_1)| < |sup(A)| + |sup(B)|$, by induction hypothesis $[\![A_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_1]\!] = [\![A_1 \setminus\!\!\setminus B_1]\!]$. Thus $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = (\{l \vee [\![A_1 \setminus\!\!\setminus B_1]\!]\} \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!]) \cup ([\![A_2]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!])$. But as for previous case, since no clause of $[\![B_2]\!]$ contains the literal l, if one of them subsumes some clause of $\{l \vee [\![A_1 \setminus\!\!\setminus B_1]\!]\}$ it must also subsume the same clause without the literal l in $[\![A_1 \setminus\!\!\setminus B_1]\!]$. Hence $\{l \vee [\![A_1 \setminus\!\!\setminus B_1]\!]\} \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!] = \{l \vee ([\![A_1 \setminus\!\!\setminus B_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!])\}$. But since $[\![A_1 \setminus\!\!\setminus B_1]\!] \subseteq [\![A_1]\!]$ we have $|sup(A_1 \setminus\!\!\setminus B_1)| \leq |sup(A_1)|$ and thus, $|sup(A_1 \setminus\!\!\setminus B_1)| + |sup(B_2)| < n$. By induction hypothesis $[\![A_1 \setminus\!\!\setminus B_1]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!] = [\![(A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2]\!]$. Similarly,

$|sup(A_2)| + |sup(B_2)| < n$ and thus, by induction hypothesis, $[\![A_2]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B_2]\!] = [\![A_2 \setminus\!\!\setminus B_2]\!]$. Hence, $[\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!] = \{l \vee [\![(A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2]\!]\} \cup [\![A_2 \setminus\!\!\setminus B_2]\!]$. This corresponds to $[\![\Delta(l, (A_1 \setminus\!\!\setminus B_1) \setminus\!\!\setminus B_2, A_2 \setminus\!\!\setminus B_2)]\!]$.

As a consequence, in all cases we have $[\![A \setminus\!\!\setminus B]\!] = [\![A]\!] \mathbin{\widehat{\setminus\!\!\setminus}} [\![B]\!]$. □

Corollary 2 If A and B encode two sets of clauses free of subsumptions, then $A \setminus\!\!\setminus B$ also encodes a set of clauses free of subsumptions.

Proof. This is an immediate consequence from the fact that $[\![A \setminus\!\!\setminus B]\!] \subseteq [\![A]\!]$. □

The purpose of our second operator is to remove subsumed clauses from a given set of clauses.

Definition 10 (Subsumption elimination) Given a set of clauses Σ, $\Sigma^\blacktriangledown$ is the set of clauses of Σ which are not subsumed by other clauses of Σ.

Property 6 Let Σ_1 and Σ_1 be two disjoint sets of clauses, $(\Sigma_1 \cup \Sigma_2)^\blacktriangledown = (\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown) \cup (\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_1^\blacktriangledown)$

Proof. By definition, any clause of $(\Sigma_1 \cup \Sigma_2)^\blacktriangledown$ is either in Σ_1 or in Σ_2, and is not subsumed by another clause of Σ_1 or of Σ_2. If it is a clause of Σ_1, it necessarily also a clause of $\Sigma_1^\blacktriangledown$. Moreover, it cannot be subsumed by a clause of Σ_2. By transitivity of subsumption, this is equivalent to say that it cannot be subsumed by a clause of $\Sigma_2^\blacktriangledown$. Thus it must be a clause of $\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown$. If it is a clause of Σ_2, we may similarly conclude that is is a clause of $\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_1^\blacktriangledown$. Thus any clause of $(\Sigma_1 \cup \Sigma_2)^\blacktriangledown$ is a clause of $(\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown) \cup (\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_1^\blacktriangledown)$. Reciprocally a clause of $\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown$ is a clause of $\Sigma_1^\blacktriangledown$ (by definition of $\widehat{\setminus\!\!\setminus}$) and thus of Σ_1. Moreover, it is not subsumed by any clause of either Σ_1 nor of Σ_2. Therefore it is a clause of $(\Sigma_1 \cup \Sigma_2)^\blacktriangledown$. By a similar reasonig, we may conclude that $\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown \subseteq (\Sigma_1 \cup \Sigma_2)^\blacktriangledown$ and thus that $(\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown) \cup (\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_1^\blacktriangledown) \subseteq (\Sigma_1 \cup \Sigma_2)^\blacktriangledown$.
□

Let us notice that this property only holds if Σ_1 and Σ_2 are disjoints. Otherwise a clause of one set could subsume the same clause in the other set and would no more belong to $(\Sigma_1^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_2^\blacktriangledown) \cup (\Sigma_2^\blacktriangledown \mathbin{\widehat{\setminus\!\!\setminus}} \Sigma_1^\blacktriangledown)$. In the following we always use this property under this assuption.

Another interesting property is the following:

Property 7 Let Σ_1 and Σ_2 be two sets of clauses, $(\Sigma_1 \cup \Sigma_2)^\blacktriangledown = (\Sigma_1^\blacktriangledown \cup \Sigma_2^\blacktriangledown)^\blacktriangledown$

Proof. This results from the transitivity of the subsumption. □

Definition 11 (*NoSub* operator) Let A be a ZBDD, $NoSub(A)$ is defined by:
T_1: $NoSub(1) = 1$

T_2: $NoSub(0) \; = \; 0$

R_1: $NoSub(\Delta(l, A, A)) \; = \; Nosub(A)$ *(node-elimination)*

R_2: if $A_1 \neq A_2$,
$NoSub(\Delta(l, A_1, A_2)) = \Delta(l,\, NoSub(A_1) \setminus\!\!\setminus NoSub(A_2),\, NoSub(A_2))$

Theorem 2 Let Δ be a ZBDD, $\llbracket NoSub(\Delta) \rrbracket = \llbracket \Delta \rrbracket^\blacktriangledown$.

Proof. The proof proceeds by induction on $n = |sup(\Delta)|$. The case $n = 0$ corresponds to the two terminal rules T_1 and T_2. In both cases $\llbracket \Delta \rrbracket$ does not contain any subsumed clauses.
Let us suppose that the property holds for any ZBDD which support size is strictly lower than n and let Δ be a ZBDD such that $|sup(\Delta)| = n$.
If Δ is of the form $\Delta(l, A, A)$ the rule R_2 applies and then $\llbracket NoSub(\Delta) \rrbracket = \llbracket Nosub(A) \rrbracket$. But by definition $\llbracket \Delta \rrbracket = \{l \vee \llbracket A \rrbracket\} \cup \llbracket A \rrbracket$. Any clause of the first set is thus subsumed by a clause of the second (this corresponds in fact to the *node elimination* rule for our ZBDD). Thus $\llbracket \Delta \rrbracket^\blacktriangledown = \llbracket A \rrbracket^\blacktriangledown$. Since $|sup(A)| < |sup(\Delta)|$, by induction hypothesis we have $\llbracket A \rrbracket^\blacktriangledown = \llbracket NoSub(A) \rrbracket$. Hence we have $\llbracket NoSub(\Delta) \rrbracket = \llbracket \Delta \rrbracket^\blacktriangledown$
If Δ is of the form $\Delta(l, A_1, A_2)$, with $A_1 \neq A_2$, the rule R_2 applies and then $\llbracket NoSub(\Delta) \rrbracket = \llbracket \Delta(l,\, NoSub(A_1) \setminus\!\!\setminus NoSub(A_2),\, NoSub(A_2)) \rrbracket$. But $(\llbracket \Delta \rrbracket^\blacktriangledown = (\{l \vee \llbracket A_1 \rrbracket\} \cup \llbracket A_2 \rrbracket)^\blacktriangledown$. Since both sets are disjoints, by property 6, it corresponds to $(\{l \vee \llbracket A_1 \rrbracket\}^\blacktriangledown \,\widehat{\setminus\!\!\setminus}\, \llbracket A_2 \rrbracket^\blacktriangledown) \cup (\llbracket A_2 \rrbracket^\blacktriangledown \,\widehat{\setminus\!\!\setminus}\, \{l \vee \llbracket A_1 \rrbracket\}^\blacktriangledown)$. We also have $\{l \vee \llbracket A_1 \rrbracket\}^\blacktriangledown = \{l \vee \llbracket A_1 \rrbracket^\blacktriangledown\}$. Since clauses of $\llbracket A_2 \rrbracket^\blacktriangledown$ do not contain the literal l, we have $\{l \vee \llbracket A_1 \rrbracket^\blacktriangledown\} \,\widehat{\setminus\!\!\setminus}\, \llbracket A_2 \rrbracket^\blacktriangledown = \{l \vee (\llbracket A_1 \rrbracket^\blacktriangledown \,\widehat{\setminus\!\!\setminus}\, \llbracket A_2 \rrbracket^\blacktriangledown)\}$ as well as $\llbracket A_2 \rrbracket^\blacktriangledown \,\widehat{\setminus\!\!\setminus}\, \{l \vee \llbracket A_1 \rrbracket\}^\blacktriangledown = \llbracket A_2 \rrbracket^\blacktriangledown$. Since the size of the support of A_1 and of A_2 is strictly smaller than n, by induction hypothesis, we know that $\llbracket NoSub(A_1) \rrbracket = \llbracket A_1 \rrbracket^\blacktriangledown$ and that $\llbracket NoSub(A_2) \rrbracket = \llbracket A_2 \rrbracket^\blacktriangledown$. Hence $\llbracket \Delta \rrbracket^\blacktriangledown = \{l \vee (\llbracket NoSub(A_1) \rrbracket \,\widehat{\setminus\!\!\setminus}\, \llbracket NoSub(A_2) \rrbracket)\} \cup \llbracket NoSub(A_2) \rrbracket$. By theorem 1, $\llbracket NoSub(A_1) \rrbracket \,\widehat{\setminus\!\!\setminus}\, \llbracket NoSub(A_2) \rrbracket = \llbracket NoSub(A_1) \setminus\!\!\setminus NoSub(A_2) \rrbracket$. Thus, $\llbracket \Delta \rrbracket^\blacktriangledown = \llbracket \Delta(l,\, NoSub(A_1) \setminus\!\!\setminus NoSub(A_2),\, NoSub(A_2)) \rrbracket$. □

Corollary 3 Let A and B be two ZBDDs,
$A \setminus\!\!\setminus B = A \setminus\!\!\setminus NoSub(B)$ and
$NoSub(A \setminus\!\!\setminus B) = NoSub(A) \setminus\!\!\setminus B = NoSub(A) \setminus\!\!\setminus NoSub(B)$

Proof. This results from the transitivity of the subsumption. □

Definition 12 ($\sqcup$ operator) Let A and B be two ZBDD, the ZBDD $A \sqcup B$ is defined by:

T_1 : $0 \sqcup A = NoSub(A)$

T_2 : $1 \sqcup A = 1$

T_{3a}: $\Delta(l, A_1, A_2) \sqcup 0 \; = \; NoSub(\Delta(l, A_1, A_2))$

T_{3b}: $\Delta(l, A_1, A_2) \sqcup 1 = 1$

R_1 : if $l < m$,
$\Delta(l, A_1, A_2) \sqcup \Delta(m, B_1, B_2) = \Delta(l,\ NoSub(A_1) \setminus\!\!\setminus (A_2 \sqcup \Delta(m, B_1, B_2)),$
$A_2 \sqcup \Delta(m, B_1, B_2))$

R_2 : if $l > m$, $\Delta(l, A_1, A_2) \sqcup \Delta(m, B_1, B_2) =$
$\Delta(m, NoSub(B_1) \setminus\!\!\setminus (B_2 \sqcup \Delta(l, A_1, A_2)),\quad B_2 \sqcup \Delta(l, A_1, A_2))$

R_3 : $\Delta(l, A_1, A_2) \sqcup \Delta(l, B_1, B_2) =$
$\Delta(l,\ (A_1 \sqcup B_1) \setminus\!\!\setminus (A_2 \sqcup B_2),\ (A_2 \sqcup B_2))$

Definition 13 (Subsumption-free union) Let Σ_1 and Σ_2 be two sets of clauses, the *subsumption free union* of Σ_1 and Σ_2 is $\Sigma_1 \widehat{\cup} \Sigma_2 = (\Sigma_1 \cup \Sigma_2)^{\blacktriangledown}$.

Theorem 3 Let A and B be ZBDDs, $[\![A \sqcup B]\!] = [\![A]\!] \widehat{\cup} [\![B]\!]$

Proof. We first show that the property holds for all pairs of ZBDD corresponding to one of the terminal cases T_1, T_2, T_{3a} and T_{3b}. If $A = 0$, $[\![A]\!] = \emptyset$, then $[\![A]\!] \widehat{\cup} [\![B]\!] = (\emptyset \cup [\![B]\!])^{\blacktriangledown}$ which corresponds to $[\![B]\!]^{\blacktriangledown}$, i.e. to $[\![NoSub(B)]\!]$ by theorem 2. This corresponds to the rule T_1. If $A = 1$, $[\![A]\!]$ is reduced to the empty clause, which subsumes all other clauses. In that case $[\![A]\!] \widehat{\cup} [\![B]\!] = \{\square\}$, which is in accordance with the rule T_2. If $A \neq 0$ and $A \neq 1$, but $B = 0$ or $B = 1$, the reasoning is symmetrical. This corresponds to the rules T_{3a} and T_{3b}.
The rest of the proof proceeds by induction on $n = |sup(A)| + |sup(B)|$. The property holds if $n = 0$, which corresponds to one of the above considered terminal cases. Let $n \geq 0$, let us suppose that the property holds for any ZBDD A and B such that $|sup(A)| + |sup(B)| < n$ and let us consider two ZBDD A and B such that $|sup(A)| + |sup(B)| = n$. The case where $|sup(A)| = 0$ corresponds to the terminal case T_1 or T_2. The case where $|sup(B)| = 0$ corresponds to the terminal case T_{3a} or T_{3b}. If both $|sup(A)| > 0$ and $|sup(B)| > 0$, A and B are respectively of the form $\Delta(l, A_1, A_2)$ and $\Delta(m, B_1, B_2)$. Three cases may then be considered.

- If $l < m$, the rule R_1 applies and $[\![A \sqcup B]\!] = [\![\Delta(l,\ NoSub(A_1) \setminus\!\!\setminus (A_2 \sqcup B), A_2 \sqcup B)]\!]$, which corresponds to $\{l \vee [\![NoSub(A_1) \setminus\!\!\setminus (A_2 \sqcup B)]\!]\} \cup [\![A_2 \sqcup B]\!]$.

  Now $[\![A]\!] \widehat{\cup} [\![B]\!] = ([\![A]\!] \cup [\![B]\!])^{\blacktriangledown} = (\{l \vee [\![A_1]\!]\} \cup [\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown}$ or $(\{l \vee [\![A_1]\!]\} \cup ([\![A_2]\!] \cup [\![B]\!]))^{\blacktriangledown}$. Since $\{l \vee [\![A_1]\!]\}$ and $[\![A_2]\!] \cup [\![B]\!]$ are disjoints, by property 6, this corresponds to $(\{l \vee [\![A_1]\!]\}^{\blacktriangledown} \widehat{\setminus\!\!\setminus} ([\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown}) \cup (([\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown} \widehat{\setminus\!\!\setminus} \{l \vee [\![A_1]\!]\}^{\blacktriangledown})$. But since no clause of $[\![A_2]\!] \cup [\![B]\!]$ contains the litteral l, this corresponds to $(\{l \vee ([\![A_1]\!]^{\blacktriangledown} \widehat{\setminus\!\!\setminus} ([\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown})\}) \cup ([\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown}$. Since $|sup(A_2)| < |sup(A)|$, we have $|sup(A_2)| + |sup(B)| < n$ and thus, by induction hypothesis, $([\![A_2]\!] \cup [\![B]\!])^{\blacktriangledown} = [\![A_2 \sqcup B]\!]$. By theorem 2, $[\![A_1]\!]^{\blacktriangledown} = [\![NoSub(A_1)]\!]$. Hence, by theorem 1, we have Thus $[\![A]\!] \widehat{\cup} [\![B]\!] = \{l \vee [\![NoSub(A_1) \setminus\!\!\setminus (A_2 \sqcup B)]\!]\} \cup [\![A_2 \sqcup B]\!]$.

- If $l > m$, the rule R_2 applies and the reasoning is similar to the case of the rule R_1.

- If $l = m$, the rule R_3 applies, and $[\![A \sqcup B]\!] = [\![\Delta(l,\, (A_1 \sqcup B_1) \mathbin{\backslash\!\!\backslash} (A_2 \sqcup B_2),\, (A_2 \sqcup B_2))]\!]$, which corresponds to $\{l \vee [\![(A_1 \sqcup B_1) \mathbin{\backslash\!\!\backslash} (A_2 \sqcup B_2)]\!]\} \cup [\![A_2 \sqcup B_2]\!]$.

  Now, $[\![A]\!] \mathbin{\hat{\cup}} [\![B]\!] = ([\![A]\!] \cup [\![B]\!])^{\blacktriangledown} = ((\{l \vee [\![A_1]\!]\} \cup [\![A_2]\!]) \cup (\{l \vee [\![B_1]\!]\} \cup [\![B_2]\!]))^{\blacktriangledown}$. By associativity and commutativity of $\cup$, this may be written as $(\{l \vee ([\![A_1]\!] \cup [\![B_1]\!])\} \cup ([\![A_2]\!] \cup [\![B_2]\!]))^{\blacktriangledown}$. Since, $(\{l \vee ([\![A_1]\!] \cup [\![B_1]\!])\}$ and$([\![A_2]\!] \cup [\![B_2]\!]))$ are disjoints, By property 6, this corresponds to $(\{l \vee ([\![A_1]\!] \cup [\![B_1]\!])\}^{\blacktriangledown} \mathbin{\widehat{\backslash\!\!\backslash}} ([\![A_2]\!] \cup [\![B_2]\!])^{\blacktriangledown}) \cup (([\![A_2]\!] \cup [\![B_2]\!])^{\blacktriangledown} \mathbin{\widehat{\backslash\!\!\backslash}} \{l \vee ([\![A_1]\!] \cup [\![B_1]\!])\}^{\blacktriangledown})$. But since no clause of $[\![A_2]\!] \cup [\![B_2]\!]$ contains the litteral l, this corresponds to $(\{l \vee (([\![A_1]\!] \cup [\![B_1]\!])^{\blacktriangledown} \mathbin{\widehat{\backslash\!\!\backslash}} ([\![A_2]\!] \cup [\![B_2]\!])^{\blacktriangledown}\}) \cup ([\![A_2]\!] \cup [\![B_2]\!])^{\blacktriangledown}$. Since $|sup(A_1)| + |sup(B_1)| < n$ and $|sup(A_2)| + |sup(B_2)| < n$, by induction hypothesis, we have $([\![A_1]\!] \cup [\![B_1]\!])^{\blacktriangledown} = [\![A_1 \sqcup B_1]\!]$ as well as $([\![A_2]\!] \cup [\![B_2]\!])^{\blacktriangledown} = [\![A_2 \sqcup B_2]\!]$. Hence, by theorem 1 it follows that $[\![A]\!] \mathbin{\hat{\cup}} [\![B]\!] = \{l \vee [\![(A_1 \sqcup B_1) \mathbin{\backslash\!\!\backslash} (A_2 \sqcup B_2)]\!]\} \cup [\![A_2 \sqcup B_2]\!]$.

  As a conclusion, we have $[\![A]\!] \mathbin{\hat{\cup}} [\![B]\!] = [\![A \sqcup B]\!]$.

□

Let us note that in the definition 12, the occurence of the $NoSub$ operator in the rule $T_1, T_{3a}, R_1 and R_2$ is used to cover the case were the initial arguments correspond to set of clauses that are not free of subsumed clauses. But if we know that the arguments encode sets of clauses which are free of subsumptions, the definition of the operation may be optimized by suppressing the calls to $NoSub$.

We now introduce our major operator. It is designed to construct the ZBDD corresponding to the distribution of two sets of clauses. For a better lisibility, and since we have to pay a particular attention to the distribution of clauses containing complementary literals, we introduce the new notation $\nabla(x, A_1, A_2, A_3)$ as a shortcut of a ZBDD of the form $\Delta(x, A_1, \Delta(\neg x, B_1, B_2))$, where x corresponds to a propositional variable. Note that, because of the tautology elimination rule, A_1 cannot be labeled by $\neg x$. Thus A_1, A_2 and A_3 do only contain nodes labeled by literals of higher order than $\neg x$.

Definition 14 ($\times$ operator) Let A and B be two ZBDD, the ZBDD $A \times B$ is defined by:

T_1 : $0 \times A = 0$

T_2 : $1 \times A = NoSub(A)$

T_{3a}: $\Delta(l, A_1, A_2) \times 0 = 0$

T_{3b}: $\Delta(l, A_1, A_2) \times 1 = NoSub(\Delta(l, A_1, A_2))$

R_1 : $\nabla(x, A_1, A_2, A_3) \times \nabla(x, B_1, B_2, B_3) =$
$\nabla(x,\, ((A_1 \times B_1) \sqcup (A_1 \times B_3) \sqcup (A_3 \times B_1)) \mathbin{\backslash\!\!\backslash} (A_3 \times B_3),$
$((A_2 \times B_2) \sqcup (A_2 \times B_3) \sqcup (A_3 \times B_2)) \mathbin{\backslash\!\!\backslash} (A_3 \times B_3),$
$(A_3 \times B_3))$

R_{2a}: if $label(A_3) \neq \neg x,\ \Delta(x, A_1, A_3)\ \times\ \nabla(x, B_1, B_2, B_3)=$
$\nabla(x,\ ((A_1 \times B_1) \sqcup (A_1 \times B_3) \sqcup (A_3 \times B_1)) \setminus\!\!\setminus (A_3 \times B_3),$
$(A_3 \times B_2) \setminus\!\!\setminus (A_3 \times B_3),\ (A_3 \times B_3))$

R_{3a}: $\Delta(\neg x, A_2, A_3)\ \times\ \nabla(x, B_1, B_2, B_3)=$
$\nabla(x,\ (A_3 \times B_1) \setminus\!\!\setminus (A_3 \times B_3),$
$((A_2 \times B_2) \sqcup (A_2 \times B_3) \sqcup (A_3 \times B_2)) \setminus\!\!\setminus (A_3 \times B_3),$
$(A_3 \times B_3))$

R_{4a}: if $label(A_3) \neq \neg x, \quad \Delta(x, A_1, A_3) \times \Delta(\neg x, B_2, B_3)=$
$\nabla(x,\ (A_1 \times B_3) \setminus\!\!\setminus (A_3 \times B_3),$
$(A_3 \times B_1) \setminus\!\!\setminus (A_3 \times B_3),\ (A_3 \times B_3))$

R_{5a}: if $label(A_3) \neq \neg x,\ label(B_3) \neq \neg x,$
$\Delta(x, A_1, A_3)\ \times\ \Delta(x, B_1, B_3)=$
$\Delta(x, ((A_1 \times B_1) \sqcup (A_3 \times B_1) \sqcup (A_1 \times B_3)) \setminus\!\!\setminus (A_3 \times B_3),$
$A_3 \times B_3)$

R_{5b}: $\Delta(\neg x, A_2, A_3)\ \times\ \Delta(\neg x, B_2, B_3)=$
$\Delta(\neg x, ((A_2 \times B_2) \sqcup (A_2 \times B_3) \sqcup (A_3 \times B_2)) \setminus\!\!\setminus (A_3 \times B_3),$
$A_3 \times B_3)$

R_{6a}: if $l \in \{x, \neg x\}$ and $\neg x < m,\ \Delta(l, A_1, A_2) \times \Delta(m, B_1, B_2)=$
$\Delta(l,\ (A_1 \times \Delta(m, B_1, B_2)) \setminus\!\!\setminus (A_2 \times \Delta(m, B_1, B_2)),$
$A_2 \times \Delta(m, B_1, B_2))$

and $R_{2b}, R_{3b}, R_{4b}, R_{6b}$ are symmetrical rules of $R_{2a}, R_{3a}, R_{4a}, R_{6a}$

Theorem 4 Let A and B be two ZBDDs, $[\![A \times B]\!]$ is the ZBDD encoding the subsumption-free distribution of $[\![A]\!]$ by $[\![B]\!]$: $[\![A \times B]\!] = ([\![A]\!] \otimes [\![B]\!])^{\blacktriangledown}$

Proof. Let A and B be two ZBDD and let us consider the subsumption free distribution of $[\![A]\!]$ over $[\![B]\!]$. We first show that the property holds for all pairs of ZBDD corrresponding to one of the terminal cases $T_1, T_2, T_{3a} and T_{3b}$. If $A = 0$, $[\![A]\!] = \emptyset$. Then $[\![A]\!] \otimes [\![B]\!] = \emptyset$ and thus $[\![A]\!] \otimes [\![B]\!]^{\blacktriangledown} = \emptyset$. This corresponds to the case T_1.
If $A = 1$, $[\![A]\!] = \{\square\}$. Then $\{\square\} \otimes [\![B]\!] = [\![B]\!]$ and $[\![A]\!] \otimes [\![B]\!]^{\blacktriangledown} = [\![B]\!]^{\blacktriangledown}$. By theorem 2, $[\![B]\!]^{\blacktriangledown} = [\![NoSub(B)]\!]$. This corresponds to the rule T_2. If $A \neq 0$ and $A \neq 1$, but $A = 0$ or $A = 1$, the reasoning is symmetrical and corresponds to the rules T_{3a} and T_{3b}.
The rest of the proof proceeds by induction on $n = |sup(A)| + |sup(B)|$. The case $n = 0$ is covered by the rules T_1 and T_2. Let us suppose that the property holds for any ZBDD A et B such that $|sup(A)| + |sup(B)| < n$ and let us consider two ZBDD A et B such that $|sup(A)| + |sup(B)| = n$. The case where $|sup(A)| = 0$ is again covered by the rules T_1 and T_2. The case where $|sup(B)| = 0$ is covered by

the rules T_{3a} and T_{3b}. Otherwise, A and B contain both at least one internal node. The eleven rules correspond to the different possible configurations.

The most general case corresponds to the situation where $A = \nabla(x, A_1, A_2, A_3)$ and $B = \nabla(x, B_1, B_2, B_3)$. Then the rule R_1 applies and

$$\begin{aligned}[\![A \times B]\!] = [\![\nabla(x, \quad & ((A_1 \times B_1) \sqcup (A_1 \times B_3) \sqcup (A_3 \times B_1)) \setminus\!\!\setminus (A_3 \times B_3), \\ & ((A_2 \times B_2) \sqcup (A_2 \times B_3) \sqcup (A_3 \times B_2)) \setminus\!\!\setminus (A_3 \times B_3), \\ & A_3 \times B_3)]\!]\end{aligned}$$

But in this case, $[\![A]\!] = \{x \vee [\![A_1]\!]\} \cup \{\neg x \vee [\![A_2]\!]\} \cup [\![A_3]\!]$ and $[\![B]\!] = \{x \vee [\![B_1]\!]\} \cup \{\neg x \vee [\![B_2]\!]\} \cup [\![B_3]\!]$. Thus

$$\begin{aligned}([\![A]\!] \otimes [\![B]\!])^{\blacktriangledown} = \quad & ((\{x \vee [\![A_1]\!]\} \cup \{\neg x \vee [\![A_2]\!]\} \cup [\![A_3]\!]) \otimes \\ & (\{x \vee [\![B_1]\!]\} \cup \{\neg x \vee [\![B_2]\!]\} \cup [\![B_3]\!]))^{\blacktriangledown}.\end{aligned}$$

Because of the distributivity of $\otimes$ with respect to $\cup$, this corresponds to

$$\begin{aligned}(\quad & (\{x \vee [\![A_1]\!]\} \otimes \{x \vee [\![B_1]\!]\}) \cup (\{x \vee [\![A_1]\!]\} \otimes \{\neg x \vee [\![B_2]\!]\}) \cup \\ & (\{x \vee [\![A_1]\!]\} \otimes [\![B_3]\!]) \cup (\{\neg x \vee [\![A_2]\!]\} \otimes \{x \vee [\![B_1]\!]\}) \cup \\ & (\{\neg x \vee [\![A_2]\!]\} \otimes \{\neg x \vee [\![B_2]\!]\}) \cup (\{\neg x \vee [\![A_2]\!]\} \otimes [\![B_3]\!]) \cup \\ & ([\![A_3]\!] \otimes \{x \vee [\![B_1]\!]\}) \cup ([\![A_3]\!] \otimes \{\neg x \vee [\![B_2]\!]\}) \cup ([\![A_3]\!] \otimes [\![B_3]\!]))^{\blacktriangledown}.\end{aligned}$$

However, clauses produced by $(\{x \vee [\![A_1]\!]\} \otimes \{\neg x \vee [\![B_2]\!]\})$ and $(\{\neg x \vee [\![A_2]\!]\} \otimes \{x \vee [\![B_1]\!]\})$ contain both literals x and $\neg x$ and thus are tautologies, which are eliminated. Among the remaining sets of clause, we may distinguish between the set of clauses containing the literal x, those containing the literal $\neg x$ and those containing neither x nor $\neg x$. Thus

$$\begin{aligned}([\![A]\!] \otimes [\![B]\!])^{\blacktriangledown} = & \{x \vee (([\![A_1]\!] \otimes [\![B_1]\!]) \cup ([\![A_1]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_1]\!]))\} \cup \\ & \{\neg x \vee (([\![A_2]\!] \otimes [\![B_2]\!]) \cup ([\![A_2]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_2]\!]))\} \cup \\ & ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown}.\end{aligned}$$

Since the tree sets are disjoints, since clauses of the first subset do not contain the literal $\neg x$, since clauses of the second subset do not contain the literal x and since clauses of the third subset do not contain either x or $\neg x$, using twice property 6 we obtain:

$$\begin{aligned}& ([\![A]\!] \otimes [\![B]\!])^{\blacktriangledown} = \\ & \quad (\{x \vee (([\![A_1]\!] \otimes [\![B_1]\!]) \cup ([\![A_1]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_1]\!]))\}^{\blacktriangledown} \\ & \qquad \widehat{\setminus\!\!\setminus} ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown}) \cup \\ & \quad (\{\neg x \vee (([\![A_2]\!] \otimes [\![B_2]\!]) \cup ([\![A_2]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_2]\!]))\}^{\blacktriangledown} \\ & \qquad \widehat{\setminus\!\!\setminus} ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown}) \cup \\ & \quad ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown}.\end{aligned}$$

Since $[\![A_3]\!]$ and $[\![B_3]\!]$ do not contain either x or $\neg x$, this corresponds to

$$\begin{aligned}& \{x \vee ((([\![A_1]\!] \otimes [\![B_1]\!]) \cup ([\![A_1]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_1]\!]))^{\blacktriangledown} \\ & \qquad \widehat{\setminus\!\!\setminus} ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown})\} \cup \\ & (\{\neg x \vee (([\![A_2]\!] \otimes [\![B_2]\!]) \cup ([\![A_2]\!] \otimes [\![B_3]\!]) \cup ([\![A_3]\!] \otimes [\![B_2]\!]))\}^{\blacktriangledown} \\ & \qquad \widehat{\setminus\!\!\setminus} ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown})\} \cup \\ & ([\![A_3]\!] \otimes [\![B_3]\!])^{\blacktriangledown}.\end{aligned}$$

By property 7, this corresponds to

$$\{x \vee (((\llbracket A_1 \rrbracket \otimes \llbracket B_1 \rrbracket)^{\blacktriangledown} \cup (\llbracket A_1 \rrbracket \otimes \llbracket B_3 \rrbracket)^{\blacktriangledown} \cup (\llbracket A_3 \rrbracket \otimes \llbracket B_1 \rrbracket)^{\blacktriangledown})^{\blacktriangledown} \\ \widehat{\setminus}\, (\llbracket A_3 \rrbracket \otimes \llbracket B_3 \rrbracket)^{\blacktriangledown})\} \cup \\ \{\neg x \vee (((\llbracket A_2 \rrbracket \otimes \llbracket B_2 \rrbracket)^{\blacktriangledown} \cup (\llbracket A_2 \rrbracket \otimes \llbracket B_3 \rrbracket)^{\blacktriangledown} \cup (\llbracket A_3 \rrbracket \otimes \llbracket B_2 \rrbracket)^{\blacktriangledown})^{\blacktriangledown} \\ \widehat{\setminus}\, (\llbracket A_3 \rrbracket \otimes \llbracket B_3 \rrbracket)^{\blacktriangledown})\} \cup \\ (\llbracket A_3 \rrbracket \otimes \llbracket B_3 \rrbracket)^{\blacktriangledown}.$$

Since $\forall i, j,\ 1 \leq i, j \leq 3, |sup(A_i)| < |sup(A)|$ and $|sup(B_j)| < |sup(B)|$ we have $\forall 1 \leq i, j \leq 3, |sup(A_i)| + |sup(B_j)| < n$. Therefore by induction hypothesis, we have $\forall 1 \leq i, j \leq 3, \llbracket A_i \times B_j \rrbracket = (\llbracket A_i \rrbracket \otimes \llbracket B_j \rrbracket)^{\blacktriangledown}$. Thus,

$$(\llbracket A \rrbracket \otimes \llbracket B \rrbracket)^{\blacktriangledown} = \\ \{x \vee (((\llbracket A_1 \times B_1 \rrbracket) \cup (\llbracket A_1 \times B_3 \rrbracket) \cup (\llbracket A_3 \times B_1 \rrbracket))^{\blacktriangledown}\, \widehat{\setminus}\, (\llbracket A_3 \times B_3 \rrbracket))\} \cup \\ \{\neg x \vee (((\llbracket A_2 \times B_2 \rrbracket) \cup (\llbracket A_2 \times B_3 \rrbracket) \cup (\llbracket A_3 \times B_2 \rrbracket))^{\blacktriangledown}\, \widehat{\setminus}\, (\llbracket A_3 \times B_3 \rrbracket))\} \cup \\ (\llbracket A_3 \times B_3 \rrbracket).$$

By theorem 3 and theorem 1, this is equivalent to

$$\{x \vee (((\llbracket A_1 \times B_1 \rrbracket) \sqcup (\llbracket A_1 \times B_3 \rrbracket) \sqcup (\llbracket A_3 \times B_1 \rrbracket)) \setminus (\llbracket A_3 \times B_3 \rrbracket))\} \cup \\ \{\neg x \vee (((\llbracket A_2 \times B_2 \rrbracket) \sqcup (\llbracket A_2 \times B_3 \rrbracket) \sqcup (\llbracket A_3 \times B_2 \rrbracket)) \setminus (\llbracket A_3 \times B_3 \rrbracket))\} \cup \\ (\llbracket A_3 \times B_3 \rrbracket).$$

This corresponds to the right side of the rule R_1.

The proof for the remaining rules may be done exactly in the same way. Each rule covers a particular case, where one, two or three components among $\{A_1, A_2, B_1, B_2\}$ are missing in the ZBDDs A and B.

The rule R_{2a} (resp. R_{2b}) covers the case where the component B_2 (resp. A_2) is missing. The rule R_{3a} (resp. R_{3b}) covers the case where the component B_1 (resp. A_1) is missing. The rule R_{4a} (resp. R_{4b}) covers the case where the component A_1 and B_2 (resp. A_2 and B_1) are missing. The rule R_{5a} (resp. R_{5b}) covers the case where the component A_2 and B_2 (resp. A_1 and B_1) are missing. The rule R_{6a} (resp. R_{6b}) covers the case where the component B_1 and B_2 (resp. A_1 and A_2) are missing. □

4. Reviving the original DP procedure

The original Davis-Putnam procedure [4], proposed in 1960, is often confused with its modified DLL version of 1962 [14]. However, the two versions differ in a fundamental way. While DLL is a backtrack search procedure for SAT, DP rather amounts to a succession of cut-elimination steps. Although generally considered as inefficient, Dechter and Rish [6] have noticed that, on some structured instances characterized by a low *induced width*, DP may perform better than DLL implementations (without backjumping and learning techniques). However, most of the time, DP produces successive *resolvents* whose size grows in a prohibitive way. Galil first proved the intractability of DP [9], using a problem for which any proof by directed resolution is exponentially long. Ten years later, Haken [10] and Urquhart [27] extended this result to all resolution based procedures. These theoretical results suggest that resolution based procedures like DP and DLL are not appropriate to solve such instances. But they rely on the implicit assumption that resolution steps are performed one by one.

```
DP($f$) :
 I.    Choose a propositional variable $x$ of $f$.
 II.   Replace all the clauses of $f$ containing $x$ (or $\neg x$)
       by those which can be obtained by resolution on $x$.
 III.  a. If the empty clause is present,
          then the original set is unsatisfiable.
       b. If the current set is empty,
          then the original set is satisfiable.
       c. Otherwise, repeat I-III for the new set of clauses.
```

Figure 4: The DP procedure

We here show that taking advantage of ZBDD structures for representing sets of clauses may lead to reconsider this point.

In [9], Galil describes the DP procedure as on figure 4. Assuming that f corresponds to a set of clauses Σ encoded by a ZBDD Δ, we now describe the whole DP algorithm in terms of ZBDD operations. The key point of the algorithm is the step II, which corresponds to the *cut elimination* of the variable x. In fact, this amounts to replace the subset Σ_x of clauses containing x and the subset $\Sigma_{\neg x}$ of clauses containing $\neg x$ by the set of clauses obtained by multiresolution from Σ_x and $\Sigma_{\neg x}$. Since multiresolution is based on clause distribution and since there is a corresponding ZBDD operator for this notion, this may be done in a rather simple way.

The ZBDDs Δ_x and $\Delta_{\neg x}$ encoding Σ_x and $\Sigma_{\neg x}$ may be obtained using standard *subset* operators [16] on ZBDD, namely $subset_1(l,\, \Delta)$, which returns the ZBDD encoding clauses containing the litteral l. Let $\Sigma_{\overline{x}}$ the set of clauses containing neither x nor $\neg x$ and $\Delta_{\overline{x}}$ be the ZBDD encoding $\Sigma_{\overline{x}}$. To obtain $\Delta_{\overline{x}}$, we introduce a new operator $without(x, \Delta)$, that may easily be implemented using the $subset_0$ operator of [16].

The next step is to caracterize the set of clauses $\Sigma_x^{'}$ and $\Sigma_{\neg x}^{'}$ obtained from Σ_x and $\Sigma_{\neg x}$ by removing the literal x from clauses of Σ_x and the literal $\neg x$ from clauses of $\Sigma_{\neg x}$. For this purpose we introduce a new simple ZBDD operator $remove(l, \Delta)$. In a ZBDD encoding a set of clauses that all contain a given litteral l, all paths to the 1 sink should pass through an internal node $\Delta(l,\, A,\, B)$ such that $B = 0$. Then, supressing the literal l from the set of clauses amounts to replace each internal node of the form $\Delta(l,\, A,\, 0)$ by A. Let $\Delta_x^{'} = remove(x, subset_1(x,\, \Delta))$ and $\Delta_{\neg x}^{'} = remove(\neg x, subset_1(\neg x,\, \Delta))$.

Since $\Sigma = \Sigma_x \cup \Sigma_{\neg x} \cup \Sigma_{\overline{x}}$, it corresponds to $= \{x \vee \Sigma_x^{'}\} \cup \{\neg x \vee \Sigma_{\neg x}^{'}\} \cup \Sigma_{\overline{x}}$ and thus to $= \{x \vee \llbracket \Delta_x^{'} \rrbracket\} \cup \{\neg x \vee \llbracket \Delta_{\neg x}^{'} \rrbracket\} \cup \llbracket \Delta_{\overline{x}} \rrbracket$. The cut elimination step then amounts to replace $\{x \vee \llbracket \Delta_x^{'} \rrbracket\}$ and $\{\neg x \vee \llbracket \Delta_{\neg x}^{'} \rrbracket\}$ by $\llbracket \Delta_x^{'} \rrbracket \otimes \llbracket \Delta_{\neg x}^{'} \rrbracket^{\blacktriangledown}$. As a consequence, thanks to theorem 4, the whole process may be fully described in terms of ZBDD operations by:

$$(\Delta_x^{'} \times \Delta_{\neg x}^{'}) \sqcup \Delta_{\overline{x}}$$

The main advantage is that the cost of this operation does not depend on the size of $\Sigma_x^{'}$, $\Sigma_{\neg x}^{'}$ and $\Sigma_{\overline{x}}$, but only on the size of $\Delta_x^{'}$, $\Delta_{\neg x}^{'}$ and $\Delta_{\overline{x}}$. As a consequence, it is possible to perform many resolutions in a single step, involving a huge number of clauses, if the corresponding ZBDDs have reasonable sizes. Using ZBDDs with such operators leads to completely reconsider the general consensus on DP inefficiency.

Another point worth reconsidering is the heuristics used for choosing the cut variable, in step I. With more conventional data structures, a traditional criterion is to select a variable which cut-elimination produces the minimal number of new clauses [2]. Of course, since the size of the ZBDD is bounded by the size of the formula, this remains an interesting idea (if not specified, this is our default heuristic). But since it is not necessarily directly related to the number of clauses, new criteria may be also be investigated.

This revived DP procedure has been implemented to evaluate the effectiveness of this approach. The resulting system, called ZRes, uses the Cudd package [25], which includes basic ZBDD operations, as well as useful dynamic reordering functions. Two versions have been realized. The standard version (*std*) uses the standard ZBDD distribution operators and deletes tautologies and subsumed clauses afterwards. The other version (*cd*, for *clause-distribution*) uses our $\times$ operator to perform the three steps simultaneously.

5. Solving two hard problems for resolution

This section presents empirical and theoretical results obtained with ZRes on two hard problems for resolution: Pigeon Hole [10] and Urquhart [27]. Since for these problems DP must generate an exponential number of clauses, they seem good candidates to evaluate the ability of our data structures to compress large sets of clauses. All tests have been performed on a Linux Pentium-II 400MHz[2] with 256MB. Our results are compared, when possible, with those of several DLL implementations: Asat [8], a good-but-simple DLL implementation, Sato 3.0 [29], which includes many optimizations such as backjumping and conflict memorization, and which, until recently, was one of those obtaining the best results on Dimacs benchmarks [7,28]. We also mention the results obtained by ZChaff [18], a recent and very much optimized version of DLL, that currently obtains the best results on many benchmark instances [28,23], as well as those of Satz-215 [12,11], that uses lookahead techniques to improve unit propagation, and those of EqSatz [13] which incorporates equivalency reasonning techniques in a DLL. All values correspond to cpu times and are expressed in seconds. We assume that an instance that cannot be solved in less than 10000 seconds counts for 10000s.

5.1. *The Pigeon Hole problem*

The well-known Pigeon Hole problem has received much attention since Haken [10] used it to prove the intractability of resolution. The simplicity of this problem,

[2]On the Dimacs [7] machine scale benchmark, this machine has a user time saving of 305%.

Instances	Asat	Sato	Zchaff	Satz-215	Eqsatz	ZRes *std*	ZRes *cd*
Hole-09	11.94	8.90	5.49	12.98	257.88	1.87	1.01
Hole-10	141.96	80.94	36.04	129.58	4191.32	3.26	1.61
Hole-11	1960.77	7373.65	242.5	1804.2	>10000	5.76	2.65
Hole-12	>10000	>10000	2452.27	>10000	–	10.18	4.06
Hole-20	–	–	–	–	–	654.8	69
Hole-30	–	–	–	–	–	>10000	1102
Hole-40	–	–	–	–	–	–	9421

Table 1: Cpu time on Hole-n instances

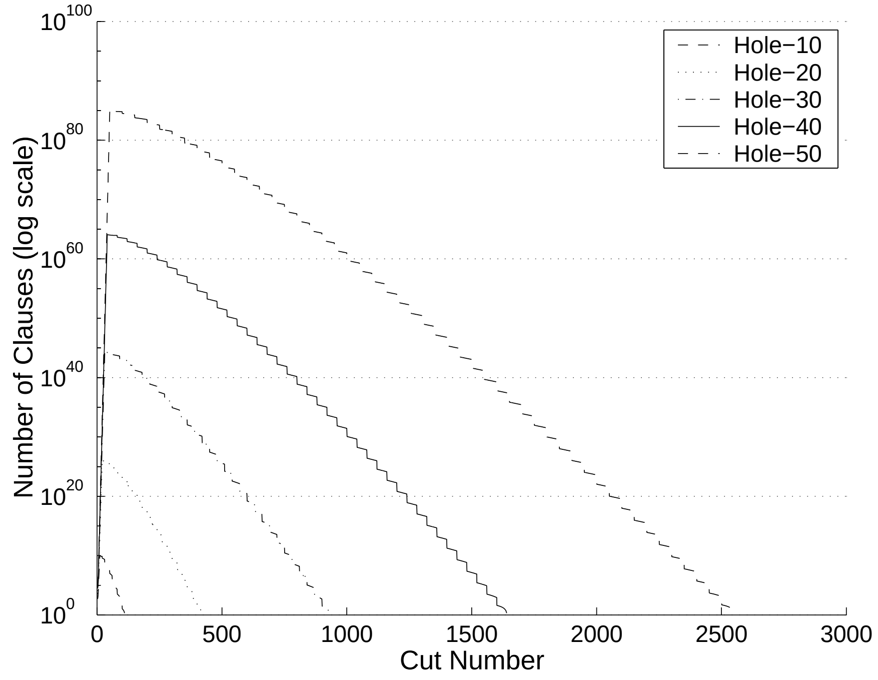
which states that $n+1$ pigeons cannot fit in n holes, contrasts with its intractability. Results of our experiments are summarized table . Surprisingly, the best results for this problem have been obtained with a heuristic function which tends to maximize the number of clauses produced at each step. This confirms that usual DP heuristics may not remain appropriate with ZBDD structures (ZRES cannot solve this problem with the default heuristics).

The first observation is that ZRes *cd* not only surpasses all DLL procedures, but also solves much bigger instances, that cannot be solved by the DLL procedures. ZRes even managed to solve Hole-55 (84756 clauses) in less than 2 days. In comparison, Hole-10 only contains 561 clauses[3]. The second observation concerns the *std* and *cd* versions of ZRes. Results confirm the effectiveness of our specialized $\times$ operator. The savings obtained with the second version even seem to increase as the size of the problem augments.

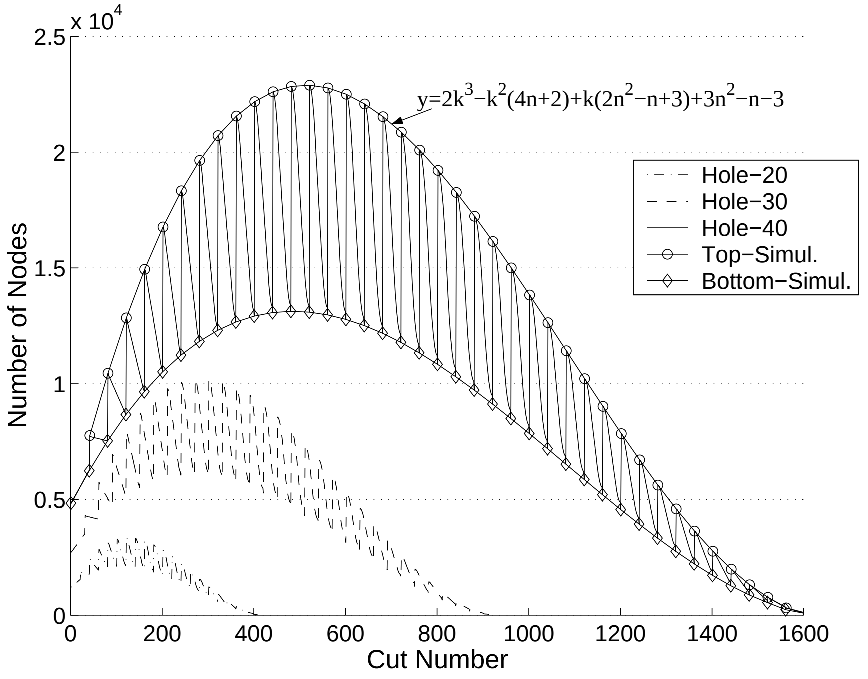
One may notice, on figure 5-a, that, from the clause point of view, the process is composed of two phases. The first phase corresponds to the first n cuts and leads to a super exponential growth. By studying precisely which cuts are performed, it can be shown that, at the end of the first phase, the total number of clauses is n^n. The second phase corresponds to a regular step by step decreasing (each of the $n+1$ steps is n cuts long). To understand how ZRes manages with such a huge number of clauses, we compare the respective variations of the total number of clauses (figure 5-a) with respect to the total number of nodes (figure 5-b). At the end of the first phase, the latter has only increased by 10%. During the second phase, the number of nodes looks like oscillating between two other curves, corresponding to the minimal and maximal values for each period (a period corresponds to n cuts). During the first cut of each period, the number of nodes jumps from its minimal to its maximal value. During the $n-1$ remaining cuts of the period, it decreases regularly until the minimal value of the next period. These oscillations clearly suggest the repetition of the same phenomenon, closely related to the structure of the formula. Indeed, sudden increases of the number of nodes correspond to eliminations of variables that break some symmetries and reduce the number of shared nodes, while the remaining cuts of a period progressively restore the regularities.

The shape of the top envelope also contrasts with the variation of the total number of clauses. It increases progressively and reaches its maximum after ap-

[3]Hole-n contains $n*(n+1)$ variables and $\frac{1}{2}(n^3+n^2)+n+1$ clauses.



a. Number of clauses



b. Number of nodes

Figure 5: Pigeon Hole resolvents

proximatively $\frac{n^2}{3}$ cuts. We have conducted an analytical study of the structure of successive ZBDDs, to evaluate the successive maximal and minimal values. The results confirm those obtained by regression analysis on experimental values and prove that the max and min envelopes correspond to polynomial functions of degree 3. On figure 5-b, we represent, for the Hole-40 case, the curve corresponding to the top and bottom envelopes and give the precise equation characterizing the top envelope obtained by analytical study, which should be read with $k = \lfloor \frac{x}{n} \rfloor$.

Theorem 5 For the Pigeon Hole problem, there exists a literal ordering and a cut-elimination ordering that lead to an exponential explosion of the number of clauses, while the number of nodes in the ZBDD encoding remains bounded by a polynomial function of degree 3.

Sketch of proof. The complete proof is too long to be reported here but may found in [22]. We only sketch here its main lines. The pigeon hole problem at the order n may be described [10] by the formula $Hole = Pos \wedge Exc$ where Pos expresses that any pigeon must be in some hole and where Exc expresses that two different pigeons cannot be in the same hole. More formally, this may be described using a set of propositional variables $\{P_{i,j}, 1 \leq i \leq n+1, 1 \leq j \leq n\}$, where $P_{i,j}$ means that *pigeon i is in hole j* and :

$$\begin{aligned} Pos &= \bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} P_{i,j} \\ Exc &= \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{l=j+1}^{n+1} \neg P_{j,i} \vee \neg P_{l,i} \end{aligned}$$

The proof strongly relies on the following variable elimination order:

$$\overbrace{P_{1,1} < \ldots < P_{1,n}}^{period\,1} < \overbrace{P_{2,1} < \ldots < P_{2,n}}^{period\,2} < \ldots < \overbrace{P_{n+1,1} < \ldots < P_{n+1,n}}^{period\,n+1}$$

As mentionned above, the whole elimination process may be decomposed into $n+1$ *periods*; The k^{th} period thus corresponds to the successive elimination of variables $P_{k,1} < P_{k,2} < \ldots < P_{k,n}$, i.e. of all variables concerning the k^{th} pigeon. Intuitively, the elimination of these variables corresponds a propagatation of constraints resulting from the fact that the k^{th} pigeon must be "placed somewhere". At the beginning of the k^{th} period (i.e. just before the cut-elimination of the variable $P_{k,1}$), all variables concerning the $k-1$ first pigeons have already been eliminated. Therefore, additional constraints have already been inferred to take into account the fact that the $k-1$ first pigeons have been placed "somewhere". The proof amounts to characterize the structure of the successive sets of clauses obtained after each elimination and to study the structure of the ZBDD encoding this set of clauses.
Particularly, it may be proved that for any $2 \leq k \leq n+1$, the formula obtained at the beginning of the k^{th} period is of the the form $Hole_k = Pos_k \wedge Exc_k \wedge Big_k$

with:

$$\begin{aligned} Pos_k &= \bigwedge_{i=k}^{n+1} \bigvee_{j=1}^{n} P_{i,j} \\ Exc_k &= \bigwedge_{j=1}^{n} \bigwedge_{i=k}^{n} \bigwedge_{l=i+1}^{n+1} \neg P_{i,j} \vee \neg P_{l,j} \\ Big_k &= \bigwedge_{S \in \mathcal{P}_{n+2-k}(n)} \bigvee_{j \in S} \bigwedge_{i=k}^{n+1} \neg P_{i,j} \end{aligned}$$

where $\mathcal{P}_j(n) = \{S \subseteq \{1, \ldots, n\} / |S| = j\}$. In this formula, Pos_k is a subset of Pos and expresses that the remaining pigeons should be in some hole, and Exc_k is a subset of Exc andexpresses that any two of the remaining pigeons cannot be in the same hole. Big_k is more tricky to understand. Intuitively, the subformula $\bigwedge_{i=k}^{n+1} \neg P_{i,j}$ expresses that none of the remaining pigeons may be placed in the hole j. If S corresponds to a subset of the holes, $\bigvee_{j \in S} \bigwedge_{i=k}^{n+1} \neg P_{i,j}$ expresses that there is one hole among S in which none of the remaining pigeons may be placed. At this stage, the last $n+2-k$ pigeons still have to be placed somewhere. Each possible mapping may be characterized by an element of $\mathcal{P}_{n+2-k}(n)$. Thus the whole Big_k expresses in some way that whatever the way the $n+2-k$ pigeons are placed into a subset of $n+2-k$ holes, there is always one of these holes in which none of the remaining pigeons may fit.
The major difficulty in this problem is to encode Big_k. Note that its above formulation is not in clausal form. But during the elimination process, it is its clausal representation that is encoded. Since there are $n+2-k$ pigeons, $(n+2-k)$ holes to be selected and C_n^{n+2-k} ways of chosing $n+2-k$ holes among n holes, Big_k corresponds in fact to to a set of $C_n^{n+2-k}.(n+2-k)^{(n+2-k)}$ clauses. For instance, at the beginning of the second period, Big_2 corresponds to n^n clauses. This explains the huge increase of the number of clauses that may be observed on figure 5-b. Still, we may prove that Big_k, using the following variable ordering:

$$P_{1,1} < P_{2,1} < \ldots < P_{n+1,1} < P_{1,2} < P_{2,2} < \ldots < P_{n+1,n}$$

may be encoded by a ZBDD the structure of which is presented on figure 6.
This ZBDD has a very regular structure. Actually each path from the top oval to the 1 sink may be viewed as a selection of $n+2-k$ holes among n, where the hole i is in the selection, if we leave the oval of line j on this path by the 1-arc. Any such path thus corresponds to some set $S \in \mathcal{P}_{n+2-k}(n)$. The whole ZBDD corresponds to all possible selections of these form (i.e. the first conjunction in Big_k). But in fact, each oval on a line j corresponds to a small ZBDD. The right part of the figure 6 corresponds to a zoomed view of any oval on line j on the left. This small set corresponds to the encoding of $\bigwedge_{i=k}^{n+1} \neg P_{i,j}$ in Big_k. We may check that the total

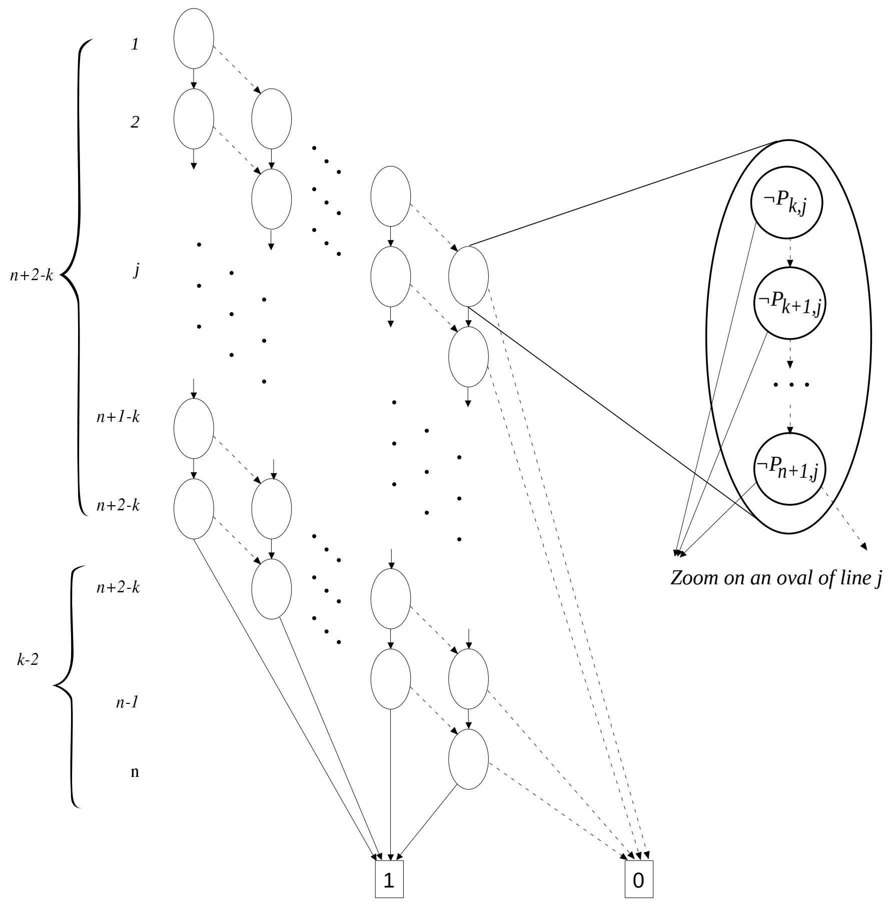


Figure 6: Pigeon Hole resolvents

number of ovals is $(n+2-k)(k-1)$ and since each oval contains $n+2-k$ nodes the total number of nodes of the ZBDD is in $O(n^3)$.
Note that $Hole_k \neq Big_k$. But it may be proved that incorporating the clauses of Pos_k and Exc_k, doesn't change the structure of the ZBDD in a fundamental way. Eventually, we have only presented the structure of $Hole_k$ at the beginning of each period k such that $2 \leq k \leq n+1$. The rest of the proof establishes that during each period, the size of the ZBDD remains in $O(n^3)$. This may be done by a precise analysis of the variations of the set of clauses, after each variable elimination of a same period, by identifying regularities in these variations and by characterizing the corresponding variations in the structure of the ZBDD encoding this set of clauses. For all details the reader is referred to [22].

□

Table 2: Cpu time on 100 Urq-3 instances

System	Total	# Solved	Mean (solved)
ASAT	404 287	69	1366
SATO	776 364	26	1398
ZRES *cd*	69.2	100	0.69

Table 3: Mean on 1000 Urquhart instances.

	Initial	Initial	Max	Max	ZRES
Urq	Var Nb	Cl Nb	Cl Nb	Nodes Nb	Time
4	78	714	10^{10}	440	1.72
6	178	1683	10^{24}	1011	8.88
8	318	3024	10^{43}	1808	29.6
10	498	4754	10^{67}	2833	72.0

As a consequence, we have shown that despite bad complexity results characterizing DP for the Pigeon Hole problem, ZRES is able to solve it effectively.

5.2. *The Urquhart problem*

The Urquhart problem [27] encodes a property of expander graphs, in a modified version of those initially proposed by Tseitin [26]. Galil [9] also used such graphs to exhibit an exponential lower bound on the cost of satifiability testing using directed resolution. Practically, Urquhart's instances correspond to a set of connected sub-bases. Each sub-base is the cnf translation of some *biconditional formula* (a chain of equivalences of the form $l_1 \leftrightarrow l_2 \leftrightarrow \ldots \leftrightarrow l_k$). As a consequence, all clauses of a given sub-base have the same length, are built using the same variables (occurring positively or negatively) and their number of negated literals are either all odd or all even. Again, we may observe on table 2 that ZRES performs very well on such examples.

We also tested bigger instances with the default heuristics. Results (table 3) clearly show that the maximum number of clauses handled in each calculus is exponentially higher than the maximum number of nodes reached. Note that here, we do not report the time of ZRES *std* time, because for this class of problem, no significant difference may be observed.

Because the Urquhart problem rather corresponds to a general framework, a precise analytical study is not possible. But it is still possible to have some intuition on the reason why ZRES still cope with such instances. By construction, each sub-base built on i variables contains 2^{i-1} clauses. But this highly structured set of clauses can be encoded by a ZBDD having only $O(i)$ nodes. Successive cuts amplify this exponential difference. As a matter of fact, a distribution between the clauses of two different sub-bases, built respectively on i and j variables, leads to 2^{i+j-2} clauses, which can be encoded with only $O(i+j)$ nodes. We thus have the following theorem:

Theorem 6 Biconditional formulæ on n variables have a cnf representation of ex-

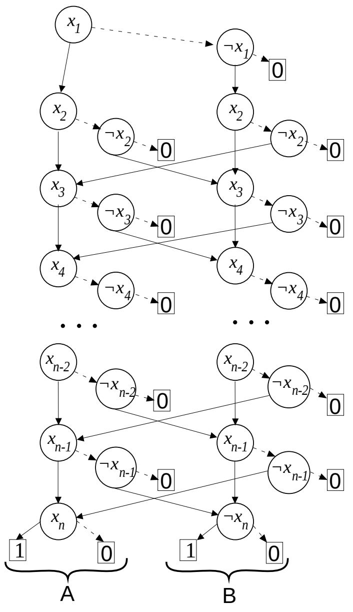


Figure 7: ZBDD encoding of an equivalency clause

ponential size $(n.2^{n-1})$ but can be encoded by a ZBDD of exactly $4.(n-1)$ nodes.

Proof. Let us recall that the set of clauses corresponding to a biconditional formula $b = l_1 \leftrightarrow l_2 \leftrightarrow \ldots \leftrightarrow l_k$ with $\forall i, 1 \leq i \leq n, l_i \in \{x_1, \neg x_i\}$is the set of clauses of $\{x_1, \neg x_1\} \otimes \{x_2, \neg x_2\} \otimes \ldots \otimes \{x_n, \neg x_n\}$ of which the number of negative literals has the same parity as $nbnl(b) + n + 1$, where $nbnl(b)$ denotes the number of negative literals in the biconditional b. Let us now present a ZBDD Δ that exactly encodes this set.

1. Create four nodes na_1, nb_1, na_n and nb_n. Label na_1 by x_1 and nb_1 by $\neg x_1$. If $nbnl(b) + n + 1$ is odd, then label na_n by $\neg x_n$ and nb_n by and x_n. Otherwise, label na_n by x_n and nb_n by and $\neg x_n$.
2. $\forall i, 2 \leq i \leq n-1$, create four nodes na_i^+, nb_i^+ labelled x_i by and na_i^-, nb_i^- labelled $\neg x_i$.
3. Connect na_n and nb_n to 1 through its 1-arc and to 0 through its 0-arc.
4. Connect na_{n-1}^+ (resp. nb_{n-1}^+) to na_n (resp. nb_n) through its 1-arc and to na_{n-1}^- (resp. nb_{n-1}^-) through its 0-arc. Connect na_{n-1}^- (resp. nb_{n-1}^-) to nb_n (resp. na_n) through its its 1-arc and to 0 through its 0-arc.
5. $\forall i, 2 \leq i \leq n-2$, connect na_i^+ (resp. nb_i^+) to na_{i+1}^+ (resp. nb_{i+1}^+) through its 1-arc and to na_i^- (reps. nb_i^-) through its 0-arc. Connect na_i^- (resp. nb_i^-) to nb_{i+1}^+ (resp. na_{i+1}^+) through its 1-arc and to 0 through its 0-arc.

Table 4: 3-SAT, 190 Cl, 44 Var, 1000 exp.

ZRes	cpu time	Max. Node Num.	Memory (MB)
std	306	5 638 848	186
cd	118	284 474	13

6. connect na_1 (resp. nb_1) to na_2^+ (resp. nb_2^+) through its its 1-arc and to nb_1 (resp. 0) through its 0-arc.

The structure of Δ is described by figure 7 (for the case where the parity is even). Note that all paths leading to the 1 sink are of length n and necessarily contain a 1-arc which parent node is labelled by either x_i or $\neg x_i$, $\forall i, 1 \leq i \leq n$. We may also notice that the nodes na and nb caracterize in fact two subgraphs A and B of Δ, which have almost the same structure (with the exception of the top and the bottom node) and are strongly connected. In fact, one may check that any path from the source node of length strictly smaller than n and which last node is in A corresponds to a sequence that contains an even number of negated literals. Inversely, if the last node is in B, it corresponds to a sequence containing an odd number of literals. Going from one side to the other amounts to add one more negated literal. On this graph, since the last node of A is labelled by a positive litteral and the last node of B is labelled by a negative litteral, all clauses have an even number of negative literals. Thus, Δ encodes exactly the set of clauses constructed on n variables, of length n and that contain an even number of negated literals. Note that in order to encode the complementary set of clauses that have an odd number of negated literals the only thing to do is to exchange the labels of the node na_n and nb_n. $\square$

Again, thanks to the high compression capabilities of ZBDDs, this second historical problem, in proving intractability of resolution, seems practically solvable using our DP implementation.

6. Other experimental results

6.1. ZRes *std vs* ZRes *cd*

To evaluate the benefit of using the $\times$ operator, we have performed an experimental comparison between ZRes *std* and ZRes *cd* on random sets of clauses. We use the fixed clause-length model [17], which is a reference problem in the literature. Two conclusions may be drawn from this experiment[4]. First, the obtained results (see table 4) confirm that ZRes is definitely not a good candidate for solving such unstructured instances. The work of [2] already pointed out that DP is not adequate for random instances. In this context, improvements due to ZBDDs are poor. Because they lack regularities, random instances are not really compressible. Nevertheless,

[4]Asat and Sato results do not appear in the table, because both solve such instances instantaneously

Table 5: Compression rates (std. dev.)

Problems	Beginning	Best
Alea-130c-30v	2.06 (*0.02*)	4.58 (*0.34*)
Alea-180c-42v	2.05 (*0.02*)	6.15 (*0.44*)
Pret150	1.88 (*0*)	1157 (*0*)
Par32-c	2.14 (*0.02*)	10^7 (*10^7*)
Ii8	2.43 (*1.09*)	10^7 (*10^7*)
Flat-50	1.76 (*0.01*)	9.61 (*2.14*)
Ssa	1.72 (*0.01*)	5.72 (*0.28*)
Massacci-R4	7.16 (*0.25*)	72.9 (*8.95*)
Aralia	1.79 (*0.28*)	10^6 (*10^7*)

we still may observe a significant difference between the two versions of ZRes. The explanation may be found in [2] which pointed out the importance of subsumption checking on such examples for DP. Actually, the use of the specialized $\times$ operator, instead of usual ZBDD operators, leads to significant savings, especially in memory.

6.2. *Compression capabilities of ZBDDs*

Using ZBDDs for representing sets of clauses is motivated by the hypothesis that structured instances should present some regularities, that should also be observable when expressed under cnf. A possible way to validate such a hypothesis is to consider the *compression rate* induced by this representation, defined as the ratio of the total number of nodes of the ZBDD encoding the cnf, with the total number of literals of the cnf. We measured the variation of this compression rate during the resolution of random generated as well as structured instances.

Experiments on 1000 3-SAT instances with 42 variables and 180 clauses show that the initial compression rate is almost always the same: 2.05. During the successive cuts, it increases up to 6.15 and then decreases again. It reaches its maximal value when handling the maximal number of clauses. It might be surprising to obtain compression rates greater than 1 on random instances. But, given the characteristics of the generated instances, there is some statistical evidence that clauses begin and/or end in the same way. During the calculus, the rate increase may be explained by the duplication of portions of clauses during the successive cuts. Thus, generated resolvents do no more have the caracteristics of the initial random instances.

For structured instances, results often differ in a significant way. Most of the time, the compression rate has a low initial value but reaches much higher values. This is the case for *ii*8, *pret*150 and *par32-c* from Dimacs benchmarks as well as for the problem of Massacci [15], concerning a cryptography application. The *ssa* and *flat* classes are however exceptions. But, when looking closely at the characteristics of these classes of problems, it is interesting to note that even if they correspond to real problems (graph coloring and circuit testing), both kinds of instances seem to incorporate a random factor in their generation process. This could be a possible explanation for such differences. We also used the Aralia benchmark [19], which is seldom used to evaluate SAT solvers. For these instances, the goal to achieve is only to eliminate a subset of the variables of the initial formula and to produce the resolvent. This task perfectly suits to ZRes. This benchmark is interesting because its instances correspond to models of physical systems. Again, the compression rates are quite different from those of random instances.

If these results confirm that ZBDDs are effective for compressing large structured sets of clauses, a high compression rate doesn't mean that ZRes is able to solve the problem efficiently. In particular, ZRes cannot achieve the resolution of *Massaci-R4*, *ii*8 and of *Parity-32*. More generally, for the Dimacs benchmarks, most efficient DLL remain quicker.

7. Conclusion

In this paper, we have proposed using ZBDDs to encode sets of clauses efficiently. We have shown that memory savings due to this encoding may become very high when considering structured instances. We have introduced specialized operators for clause set handling and shown that the $\times$ operator may distribute sets of clause of exponential size, while handling polynomial size data structures. This makes it possible to reconsider the original DP procedure and to propose a revised version, which is able to perform multi-resolution on large sets of clauses, in an efficient manner. This allows this new DP to solve hard instances for resolution, which contrasts with the widespread idea concerning its inefficiency.

Exponential lower bounds on the length of proofs have often led to consider resolution based procedures as intrinsically limited. It is often suggested that the way to surpass such limitations, which occur even on simple problems like Pigeon Hole, is to switch to more powerful proof systems. Our feeling is that, in some cases, the length of a proof is not a good indicator of its complexity, as only the length of a message is not significant in information theory.

Many directions are interesting for further work. We have used our $\times$ operator at the heart of a SAT prover. The introduction of ZBDDs give the opportunity to consider the DP algorithm from a different point of view. New heuristics, based on the ZBDD structure, instead of the number of clauses, have to be developed. This $\times$ operator can also be used to solve efficiently other problems such as computing sets of prime implicates [24]. From the SAT viewpoint, we think that the DLL procedure could also take advantage of ZBDD representation to handle very large sets of clauses in a compact way. A further perspective could be to develop a hybrid algorithm, merging DP and DLL, to benefit from the best of both approaches.

References

[1] R.E. Bryant. Graph - based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[2] Ph. Chatalic and L. Simon. Davis and Putnam 40 years later: a first experimentation. Technical Report 1237, LRI, Orsay, France, 2000.

[3] O. Coudert and J.-C. Madre. A new method to compute prime and essential prime implicants of boolean functions". In T. Knight and J. Savage, editors, *Advanced Research in VLSI and Parallel Systems*, pages 113–128, March 1992.

[4] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, pages 201–215, 1960.

[5] J. de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI'92*, pages 780–785, 1992.

[6] R. Dechter and I. Rish. Directional resolution: The Davis-Putnam procedure, revisited. In *Proceedings of KR-94*, pages 134–145, 1994.

[7] The DIMACS challenge benchmarks. ftp://ftp.rutgers.dimacs.edu/challenges/sat.

[8] Olivier Dubois. Can a very simple algorithm be efficient for SAT? Electronically available on ftp://ftp.dimacs.rutgers.edu/pub/challenges/sat/contributed/dubois.

[9] Zvi Galil. On the complexity of regular resolution and the Davis-Putnam procedure. *Theorical Computer Science*, 4:23–46, 1977.

[10] A. Haken. The intractability of resolution. *Theorical Computer Science*, 39:297–308, 1985.

[11] D. Le Berre. Exploiting the real power of unit propagation lookahead. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9, Boston University, Massachusetts, USA, June 14th-15th 2001. Elsevier Science Publishers. Proceedings of the LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT2001).

[12] C.-M. Li. A constrained based approach to narrow search trees for satisfiability. *Information processing letters*, 71:75–80, 1999.

[13] C.-M. Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'00)*, pages 291–296, 2000.

[14] G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, pages 394–397, 1962.

[15] F. Massacci and L. Marraro. Logical cryptanalysis as a sat-problem: Encoding and analysis of the u.s. data encryption standard. *JAR*, 2000.

[16] S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *30th ACM/IEEE Design Automation Conference*, 1993.

[17] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of sat problems. *AAAI'92*, pages 459–465, 1992.

[18] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, Juin 2001.

[19] A. Rauzy. Boolean models for reliability analysis: a benchmark. Technical report, LaBRI, Université Bordeaux I.

[20] A. Rauzy. Mathematical foundations of minimal cutsets. *IEEE Transaction on Reliability.* (to appear).

[21] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *AAAI-92*, pages 440–446, 1992.

[22] L. Simon. *Multiresolution for consistancy checking and deduction in propositional logic.* PhD thesis, Université Orsay Paris XI, France, 2001. To appear, in French.

[23] L. Simon and P. Chatalic. SATEx: a Web-based Framework for SAT Experimentation. In Henry Kautz and Bart Selman, editors, *Electronic Notes in Discrete Mathematics*, volume 9, Boston University, Massachusetts, USA, June 14th-15th 2001. Elsevier Science Publishers. Proceedings of the LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT2001).

[24] L. Simon and A. del Val. Efficient consequence finding. In *17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, pages 359–365, Seattle, Washington, USA, 2001.

[25] F. Somenzy. Cudd release 2.3.0. http://bessie.colorado.edu/˜fabio.

[26] G. Tseitin. On the complexity of derivation in propositional calculus. *studies in Constructive Mathematics and Mathematical Logic, part 2*, pages 115–125, 1968.

[27] A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34:209–219, 1987.

[28] The SatEx web site. http://www.lri.fr/˜simon/satex/satex.php3. (gathering of SAT experimentations).

[29] Hantao Zhang. SATO: An efficient propositional prover. In *CADE-14*, LNCS 1249, pages 272–275, 1997.