SAT-solving Based on Boundary Point Elimination*

Eugene Goldberg, Panagiotis Manolios

Northeastern University, USA {eigold,pete}@ccs.neu.edu

Abstract. We study the problem of building structure-aware SAT-solvers based on resolution. In this study, we use the idea of treating a resolution proof as a process of Boundary Point Elimination (BPE). We identify two problems of using SAT-algorithms with Conflict Driven Clause Learning (CDCL) for structure-aware SAT-solving. We introduce a template of resolution based SAT-solvers called BPE-SAT that is based on a few generic implications of the BPE concept. BPE-SAT can be viewed as a generalization of CDCL SAT-solvers and is meant for building new structure-aware SAT-algorithms. We give experimental results substantiating the ideas of the BPE approach. In particular, to show the importance of structural information we compare an implementation of BPE-SAT and state-of-the-art SAT-solvers on narrow CNF formulas.

1 Introduction

In the last decade, SAT-algorithms have seen a great deal of success in solving real-life formulas. (Since the key idea of modern SAT-algorithms is Conflict Driven Clause Learning we will refer to them as CDCL SAT-solvers.) This success can be attributed (among other things) to using effective heuristics controlling various aspects of the behavior of a CDCL SAT-solver such as decision making, removal of obsolete conflict clauses and so on. The problem, however, is that modern SAT-solvers are based on the resolution proof system that is most likely non-automatizable [1]. Informally, this means that there is no efficient deterministic procedure for finding short resolution proofs. So even if a class of formulas has short proofs, a *deterministic* SAT algorithm may not be able to find them and hence may have very poor performance. For example, CDCL SAT-solvers often display abysmal performance on formulas describing equivalence checking of two identical copies of combinational circuits despite the existence of linear size resolution proofs for such formulas.

A natural way to cope with non-automatizability of resolution is to provide a SAT-solver with additional information about the formula structure. In this paper, we study the problem of building structure-aware resolution-based SAT-solvers. We use the approach of [8] where a resolution proof is considered as a process of Boundary Point Elimination (BPE for short). Our intent here is to

* This research was funded in part by the SRC contract 2008-TJ-1852.

point out the problems with building structure-aware CDCL SAT-solvers and suggest a direction for future research.

Given a CNF formula F, an l-boundary point is an unsatisfying complete assignment $\boldsymbol{p}$ such that all clauses of F falsified by $\boldsymbol{p}$ have literal l. The elimination of $\boldsymbol{p}$ as a boundary point is to add a resolvent C of clauses of F such that $\boldsymbol{p}$ is not a boundary point of $F \wedge C$. Importantly, in any resolution proof, all boundary points of F have to be eliminated. An l-boundary point (where l is a literal of variable x_i) is eliminated only by particular resolutions on x_i. So boundary points of F specify "mandatory" resolutions.

We describe a generic SAT-solver called BPE-SAT that is based on the following four observations. First, *any* resolution-based SAT-algorithm has to generate mandatory resolutions. Second, it is preferable to find such resolutions early since they may make many non-mandatory resolutions redundant, which allows one to find a proof faster and make it smaller. (In particular, an entire proof may consist only of mandatory resolutions e.g. for equivalence checking formulas.) Third, finding an l-boundary point of F is hard (it reduces to a SAT-check for the formula $F \setminus \{$the clauses of F with literal $l\}$). Fourth, if $\boldsymbol{p}$ is an l-boundary point of F, F' is a subset of clauses of F and $F'(\boldsymbol{p})$=0, then $\boldsymbol{p}$ is an l-boundary point of F'. So by eliminating l-boundary points of F' one also eliminates l-boundary points of F (and if F' is small, finding boundary points of F' is much easier than for F).

BPE-SAT repetitively extracts a subformula $F'(X_1, X_2)$ of $F(X)$ (where X_1,X_2 are non-overlapping subsets of the set of variables X) and then existentially quantifies away the variables of X_1 producing clauses depending on variables of X_2. Existential quantification of F' is performed by an algorithm called the BPE procedure. To quantify away a variable $x_i \in X_1$, the BPE procedure adds to F' resolvents that eliminate all symmetric x_i-boundary and $\overline{x}_i$-boundary points. The BPE procedure eliminates variables of X_1 one by one and so works similarly to the DP algorithm [6]. However, we believe that relaxing the order in which boundary points are eliminated may lead to developing a much more powerful procedure of existential quantification.

BPE-SAT is based on a few very general observations listed above and so, is formulated as a *template* of a resolution based SAT-solver. In particular, we do not specify in BPE-SAT how a subformula F' is extracted from F. This is done for the following reason. According to the observations above, the choice of subformula F' is good if the latter shares boundary points with F. Intuitively, the complexity of identifying good subformulas F' is what makes resolution non-automatizable (the choice of F' strongly depends on the structure of F). So, in this paper, we rather formulate the problem of finding subformulas F' that share boundary points with F than solve it. Interestingly, our definition of good subformulas gives a new meaning to the notion of the formula structure in the context of resolution proofs. (The structure of formula F is specified by a set of its subformulas sharing boundary points with F).

A natural question is how BPE-SAT is related to CDCL SAT-solvers. Remarkably, despite BPE-SAT being just a theoretical construction, a CDCL SAT-solver is one of (many) implementations of BPE-SAT. The set of clauses responsible for a conflict specifies the subformula F' to be selected from F. Existential quantification of variables of F' is performed by the conflict clause generation procedure. The latter can be represented as the BPE procedure eliminating boundary points specified by the partial assignment leading to the conflict.

Another important issue is whether BPE-SAT brings in something new with respect to CDCL SAT-solvers. Both BPE-SAT and CDCL SAT-solvers are based on resolution and it was recently shown that resolution proofs can be polynomially simulated by non-deterministic CDCL SAT-solvers [9]. It is tempting to use these results for optimistic predictions about the performance of *deterministic* CDCL SAT-solvers. However, there are at least two problems that dampen this optimism. (The introduction of BPE-SAT is a way to address these problems.) The first problem is that eager backtracking of a CDCL SAT-solver (i.e. after a conflict occurs) may lead to generation of subformula F' (specified by the conflict) that does not share boundary points with F. This issue is addressed in BPE-SAT by posing the problem of finding subformulas F' sharing boundary points with F. The second problem is that a SAT-solver based only on conflict driven learning cannot efficiently quantify away the variables of X_1 from subformula $F'(X_1, X_2)$ (even if the resulting set of clauses depending on variables of X_2 is small). This problem is totally ignored by the results of [9] because it is caused by the difference in termination conditions of non-deterministic and deterministic CDCL SAT-solvers when performing existential quantification. (And [9] is concerned only with non-deterministic CDCL SAT-solvers.) In BPE-SAT, this problem is addressed by showing the promise of performing existential quantification using boundary point elimination.

The contributions of this paper are as follows. First, we introduce a template SAT-solver BPE-SAT and justify this introduction in terms of BPE. Second, we identify some problems of CDCL SAT-solvers in building structure-aware SAT-solvers and point out directions for solving those problems. In particular, we show that BPE can be used for existential quantification. Third, we give an example of tuning BPE-SAT to a particular class of formulas (narrow formulas).

2 Basic Definitions

In this section, we recall some basic definitions of SAT solving.

Definition 1. ***A literal*** *$l(x_i)$ of a Boolean variable x_i is either x_i itself or its negation (denoted as $\overline{x}_i$). We will also denote a literal as just l in the contexts where the identification of x_i can be omitted.* ***A clause*** *is the disjunction of literals where no two (or more) literals of the same variable can appear.* ***A CNF formula*** *is the conjunction of clauses. We will also view a CNF formula* ***as a set of clauses****. Denote by* ***Vars(F)*** *(respectively* ***Vars(C)****) the set of variables of CNF formula F (respectively clause C).*

Definition 2. *Given a CNF formula $F(x_1, \ldots, x_n)$,* ***a complete assignment p*** *(also called* ***a point****) is a mapping $\{x_1, \ldots, x_n\} \rightarrow \{0, 1\}$. A clause C is* ***satisfied*** *(respectively* ***falsified****) by* $\boldsymbol{p}$ *if $C(\boldsymbol{p}) = 1$ (respectively $C(\boldsymbol{p}) = 0$).*

Definition 3. *Given a CNF formula F, **a satisfying assignment p** is a complete assignment satisfying every clause of F. **The satisfiability problem** (SAT) is to find a satisfying assignment for F or to prove that such an assignment does not exist.*

Definition 4. *Let clauses C', C'' have the opposite literals of variable x_i (and only of x_i). The **resolvent** C of C' and C'' on variable x_i is the clause with all the literals of C' and C'' but those of x_i. The clause C is said to be obtained by a **resolution operation** on x_i. C' and C'' are called the **parent clauses** of C.*

Definition 5. *([2]) Let F be an unsatisfiable formula. Let $R_1, \ldots, R_k$ be a set of clauses such that*

- *each clause R_i is obtained by a resolution operation where a parent clause is either a clause of F or R_j, $j < i$;*
- *clauses R_i are numbered in the derivation order and R_k is an empty clause.*

*Then the k resolutions that produced $R_1, \ldots, R_k$ are called a **resolution proof**.*

3 Boundary Points and Resolution

The objective of this section is to discuss the relation between a resolution proof and boundary point elimination. In subsection 3.1, we define the notion of boundary points and list some of their properties. In subsection 3.2, we recall the approach of [8] that views a resolution proof as a boundary point elimination procedure.

3.1 Boundary points and their properties

Definition 6. *Denote by **Fls(p,F)** the set of clauses of a CNF formula F falsified by a complete assignment **p**.*

Definition 7. *Given a CNF formula F, a complete assignment **p** is called an l-boundary point, if Fls(**p**,F) $\neq \emptyset$ and every clause of Fls(**p**,F) contains literal l.*

Example 1. Let F consist of 5 clauses: $C_1 = x_2$, $C_2 = \overline{x}_2 \vee x_3$, $C_3 = \overline{x}_1 \vee \overline{x}_3$, $C_4 = x_1 \vee \overline{x}_3, C_5 = \overline{x}_2 \vee \overline{x}_3$. Complete assignment $\boldsymbol{p_1}$=(x_1=0,x_2=0,x_3=1) falsifies only clauses C_1,C_4. So *Fls*($\boldsymbol{p_1}$,F)={C_1,C_4}. There is no literal shared by all clauses of *Fls*($\boldsymbol{p_1}$,F). Hence $\boldsymbol{p_1}$ is not a boundary point. On the other hand, $\boldsymbol{p_2}$= (x_1=0,x_2=1,x_3=1) falsifies only clauses C_4,C_5 that share literal $\overline{x}_3$. So $\boldsymbol{p_2}$ is a $\overline{x}_3$-boundary point.

Definition 8. *Denote by **Bnd_pnts(F)** the set of all boundary points of a CNF formula F. We assume that an l-boundary point **p** is specified in Bnd_pnts(F) as the pair (l,**p**). So the same point **p** may be present in Bnd_pnts(F) more than once (e.g. if **p** is both an $l(x_i)$-boundary point and an $l(x_j)$-boundary point).*

Definition 9. *Let* $\boldsymbol{p}$ *be a complete assignment. Denote by* $flip(\boldsymbol{p},x_i)$ *the point obtained from* $\boldsymbol{p}$ *by flipping the value of* x_i*.*

The following proposition explains why studying boundary points is important.

Proposition 1. *([8]) If* F *contains at least one clause and* $Bnd\_pnts(F) = \emptyset$*, then* F *is unsatisfiable.*

Proposition 1 implies that for a satisfiable formula F, $Bnd\_pnts(F) \neq \emptyset$. In particular, it is not hard to show [8] that if $F(\boldsymbol{p'})$=0, $F(\boldsymbol{p''})$=1 and $\boldsymbol{p''}= flip(\boldsymbol{p'},x_i)$, then $\boldsymbol{p'}$ is an $l(x_i)$-boundary point. (This explains the name "boundary point".) Another interesting fact is that if $\boldsymbol{p'}$ is an x_i-boundary point, then $\boldsymbol{p''}= flip(\boldsymbol{p'},x_i)$ is either a satisfying assignment or a $\overline{x}_i$-boundary point [8]. So, for an unsatisfiable formula, all boundary points come in pairs. We will refer to such a pair of points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ as ***symmetric*** x_i-boundary and $\overline{x}_i$-boundary points.

3.2 Elimination of boundary points by adding resolvents

Let $\boldsymbol{p'}$ and $\boldsymbol{p''}$ be symmetric x_i-boundary and $\overline{x}_i$-boundary points. It is not hard to show [8] that a clause C' falsified by $\boldsymbol{p'}$ can be resolved on variable x_i with a clause C'' falsified by $\boldsymbol{p''}$. (Since points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ are x_i-boundary and $\overline{x}_i$-boundary respectively, C' and C'' have literals x_i and $\overline{x}_i$ respectively. Since $\boldsymbol{p'}$, $\boldsymbol{p''}$ are different only in the value of x_i, the clauses C' and C'' cannot have opposite literals of $x_j, j \neq i$.)

The resolution of C' and C'' above produces a clause that is falsified by both $\boldsymbol{p'}$ and $\boldsymbol{p''}$ and does not have variable x_i. Then $\boldsymbol{p'}$ and $\boldsymbol{p''}$ are not $l(x_i)$- boundary points of $F \wedge C$. So $Bnd\_pnts(F \wedge C) \subset Bnd\_pnts(F)$. (Adding a clause to F can only *eliminate* some boundary points but cannot *produce* new ones). We will refer to this process of removing boundary points by adding clauses implied by F as *boundary point elimination.* (Note that adding C to F may also eliminate a large number of $l(x_i)$-boundary points different from $\boldsymbol{p'}$ and $\boldsymbol{p''}$.)

Only symmetric x_i-boundary and $\overline{x}_i$-boundary points can be eliminated by adding a resolvent on x_i. If formula F is satisfiable (and has at least one clause), there is always an $l(x_i)$-boundary point $\boldsymbol{q'}$ such that the point $\boldsymbol{q''}$=$flip(\boldsymbol{q'},x_i)$ satisfies F [8]. Let C' be a clause of F falsified by $\boldsymbol{q'}$ (and so having literal $l(x_i)$). The resolvent C of C' with any clause C'' of F having literal $\overline{l(x_i)}$ is satisfied by $\boldsymbol{q''}$ and hence by $\boldsymbol{q'}$ (because C does not depend on x_i). So $\boldsymbol{q'}$ cannot be eliminated by adding a resolvent on variable x_i.

Let $R_1, \ldots, R_k$ be a resolution proof where R_k is an empty clause. Note that $Bnd\_pnts(F \wedge R_k) = \emptyset$. (Indeed, by definition, if $\boldsymbol{p}$ is an $l(x_i)$-boundary point, every clause falsified by $\boldsymbol{p}$ has to have at least one literal i.e. $l(x_i)$.) This means that every $l(x_i)$-boundary point $\boldsymbol{p}$ of the initial formula F is eventually eliminated. Then there is a resolvent R_{m+1} such that $\boldsymbol{p}$ is an $l(x_i)$-boundary point for $F \wedge R_1 \wedge \ldots \wedge R_m$ but not for $F \wedge R_1 \wedge \ldots \wedge R_{m+1}$. It was shown in [8] that R_{m+1} has to be obtained by a resolution on variable x_i . In other words, an $l(x_i)$-boundary point *mandates* a resolution on variable x_i.

4 *BPE-SAT*

In this section, we describe BPE-SAT, a template SAT-solver based on the idea of Boundary Point Elimination (BPE for short).

4.1 Justification of BPE-SAT

BPE-SAT is based on a few very general observations we listed in the introduction. In particular, we explained why it is important to identify mandatory resolutions, i.e. ones eliminating boundary points of F, early. (Adding mandatory resolutions allows the SAT-solver to avoid generating many "non-mandatory" resolutions.) Finding a boundary point of F is hard. If $\boldsymbol{p}$ is an l-boundary point it satisfies $F \setminus \{\text{the clauses of } F \text{ with literal } l\}$. So finding a boundary point of F reduces to a SAT-check for a formula that is not much smaller than F. BPE-SAT addresses this problem by using a conservative approach where boundary points are eliminated from subformulas F' of F. This approach is based on the following simple proposition.

Proposition 2. *Let $\boldsymbol{p}$ be an l-boundary of F and F' be a subset of clauses of F such that $F'(\boldsymbol{p}) = 0$. Then $\boldsymbol{p}$ is an l-boundary point of F'.*

Proof. The set of clauses of F' falsified by $\boldsymbol{p}$ is a non-empty subset of the set of clauses of F falsified by $\boldsymbol{p}$.

The proposition above implies that by eliminating boundary points from F' one may also eliminate boundary points from F. If F' is much smaller than F, finding boundary points of F' is much easier than those of F. From the viewpoint of BPE-SAT, a subformula F' of F is good if elimination of l-boundary points of F' also eliminates l-boundary points of F. Intuitively, finding such subformulas requires the knowledge of the structure of F. So, BPE-SAT does not specify how subformulas are chosen, leaving this work to implementations aimed at particular classes of formulas.

4.2 High-level view of BPE-SAT

```
BPE-SAT($F$)
{while ($TRUE$)
  {$F'(X_1, X_2) = pick\_subform(F)$;
  $F''(X_2) = BPE(F', X_1)$;
  if ($contains\_empty\_clause(F'')$)
    return($UNSAT$);
  $F = F \cup F''$; }}
```

Fig. 1. *BPE-SAT*

The pseudocode of BPE-SAT is shown in Figure 1. BPE-SAT iteratively performs the following three actions. First, it picks a subformula $F'(X_1, X_2)$ of the current formula F. Then it calls the BPE procedure that, for each variable $x_i \in X_1$, eliminates all symmetric x_i-boundary and $\overline{x}_i$-boundary points from F'. If the set of clauses F'' generated to eliminate boundary points from F' contains an empty clause, BPE-SAT reports that F is unsatisfiable. Otherwise, it adds the clauses of F'' to F and starts a new iteration.

In this paper, we limit the discussion of BPE-SAT to the case of unsatisfiable formulas. (The satisfiability of F can be established (a) when looking for subformula F', as it is done in CDCL SAT-solvers where F' is a set of clauses responsible for a conflict; or (b) when eliminating boundary points of F', as it is done in the implementation of BPE-SAT meant for solving narrow formulas.)

```
$BPE(G(X_1, X_2), X_1)$
{while $((X_1 \neq \emptyset)$ and $(G \neq \emptyset))$
  {$x_i = pick\_variable(X_1)$;
  $G' = elim\_bnd\_pnts(G,\, x_i)$;
  if $(contains\_empty\_clause(G'))$
    return($G'$);
  $G = (G \setminus (G_{x_i} \cup G_{\overline{x}_i})) \cup G'$;
  $X_1 = X_1 \setminus \{x_i\}$; }
return($G$);}
```

Fig. 2. *BPE* procedure

We also do not discuss the convergence of BPE-SAT (i.e. lack of looping). Typically, the convergence of a SAT-solver with learning is achieved by avoiding the generation of the same clause more than once. An analogous approach can be used in implementations of BPE-SAT but this requires a more detailed description of the choice of subformulas F'. The convergence of our implementation of BPE-SAT for narrow formulas follows from the semantics of the BPE procedure (that monotonically eliminates boundary points) and from the way subformulas F' are chosen (the choice of F' respects a fixed variable order, see Section 7).

4.3 The BPE procedure

The BPE procedure is described in Figure 2. It accepts a CNF formula $G(X_1, X_2)$ and processes the variables of X_1 in the "while" loop in a specified order. After a variable x_i of X_1 is selected, *elim_bnd_pnts* procedure is called that eliminates all symmetric x_i-boundary and $\overline{x}_i$- boundary points from G. It returns the set of clauses G' generated to eliminate such boundary points.

On termination of *elim_bnd_pnts*, G is modified as follows. The clauses of G with literals x_i and $\overline{x}_i$ (denoted as G_{x_i} and $G_{\overline{x}_i}$ respectively) are removed from G. The clauses G' generated by *elim_bnd_pnts* are added to G. As we show in Subsection 5, $(G \setminus (G_{x_i} \cup G_{\overline{x}_i})) \cup G' = G(\ldots, x_i = 0, \ldots) \vee G(\ldots, x_i = 1, \ldots)$. That is elimination of all symmetric x_i-boundary and $\overline{x}_i$- boundary points from G is equivalent to quantifying away variable x_i. Finally the variable x_i is removed from the set X_1 and a new iteration starts.

The BPE procedure terminates if 1) G does not have any clauses left or 2) X_1 is empty or 3) an empty clause was derived by the *elim_bnd_pnts* procedure. On termination, the BPE procedure returns the current formula G (that depends only on variables of X_2 due to $X_1 = \emptyset$).

4.4 Description of *elim_bnd_pnts* procedure

Given a CNF formula G and variable x_i, the *elim_bnd_pnts* procedure (shown in Figure 3) returns a set of resolvents G' eliminating all symmetric x_i-boundary and $\overline{x}_i$-boundary points of G for $x_i \in X_1$. (That is, if an $l(x_i)$-boundary point $\boldsymbol{p}$ is left in $G \wedge G'$, then $\boldsymbol{p}^* = flip(\boldsymbol{p}, x_i)$ satisfies G and so $\boldsymbol{p}$ cannot be eliminated.)

Boundary points are generated in the "while" loop. Every time a new $l(x_i)$-boundary point $\boldsymbol{p}$ is found, it is eliminated by adding a resolvent on variable x_i that is falsified by $\boldsymbol{p}$. The procedure terminates when all symmetric x_i-boundary and $\overline{x}_i$-boundary points are eliminated. Below, we describe *elim_bnd_pnts* in more detail.

```
$elim\_bnd\_pnts(G, x_i)$
$\{H = (G \setminus (G_{x_i} \cup G_{\overline{x}_i}))$;
$H = H \cup$ $dir\_clauses(G_{x_i}, G_{\overline{x}_i})$;
$G' = \emptyset$;
while ($TRUE$)
  $\{(\boldsymbol{p_0},\boldsymbol{p_1},sat) = find\_bnd\_pnts(H)$;
  if ($sat == FALSE$) return($G'$);
  $G'_{x_i} = find\_fls\_cls(\boldsymbol{p_0},G_{x_i})$;
  $G'_{\overline{x}_i} = find\_fls\_cls(\boldsymbol{p_1},G_{\overline{x}_i})$;
  $R = gen\_resolvent(G'_{x_i},G'_{\overline{x}_i})$;
  $H = H \cup \{R\}$;
  $G' = G' \cup \{R\};\}$
  return($G'$);}
```

Fig. 3. *elim_bnd_pnts* procedure

Following the definition of an $l(x_i)$-boundary point, *elim_bnd_pnts* looks for an assignment satisfying formula $H = G \setminus (G_{x_i} \cup G_{\overline{x}_i})$. Adding *directing clauses* to H (constructed by function *dir_clauses*) is meant to avoid generation of $l(x_i)$-boundary points that cannot be eliminated. This is achieved by avoiding assignments satisfying G_{x_i} and/or $G_{\overline{x}_i}$. A more detailed description of generation of directing clauses is given at the bottom of this subsection.)

The *find_bnd_pnts* procedure called in the "while" loop is a CDCL SAT-solver. It returns a pair of assignments $\boldsymbol{p_0}$,$\boldsymbol{p_1}$ symmetric in x_i that satisfy H (by construction, H does not depend on x_i) or reports that H is unsatisfiable. For the sake of clarity, we assume that $x_i = 0$ (respectively $x_i = 1$) in $\boldsymbol{p_0}$ (respectively $\boldsymbol{p_1}$). Then $\boldsymbol{p_0}, \boldsymbol{p_1}$ are x_i-boundary and $\overline{x}_i$-boundary points of G respectively.

If H is unsatisfiable, all symmetric x_i-boundary and $\overline{x}_i$-boundary points have been eliminated and *elim_bnd_pnts* terminates returning the set G' of generated resolvents. Otherwise, the clauses $G'_{x_i} \subseteq G_{x_i}$ (respectively $G'_{\overline{x}_i} \subseteq G_{\overline{x}_i}$) are generated that are falsified by $\boldsymbol{p_0}$ (respectively $\boldsymbol{p_1}$). Note that due to adding to H directing clauses, the sets G'_{x_i} and $G'_{\overline{x}_i}$ cannot be empty. Boundary points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ can be eliminated by resolving any clause of G'_{x_i} with any clause of $G'_{\overline{x}_i}$.

The choice of the resolvent eliminating boundary points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ is done by the procedure *gen_resolvent*. The latter uses a heuristic illustrated by the following example. Suppose boundary points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ are found by the BPE procedure when processing variable x_5. Let $R' = x_{10} \vee x_{20}$ and $R'' = x_{20} \vee \overline{x}_{35} \vee x_{40}$ be the two clauses obtained by resolving clauses of G'_{x_5} and $G'_{\overline{x}_5}$ (and so eliminating $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ as $l(x_5)$-boundary points). Given a clause C, denote by *min_index*(C) the minimum variable index among *Vars*(C). Hence *min_index*$(R') = 10$ and *min_index*(R'')=20. The *gen_resolvent* procedure selects the clause with the largest value of *min_index* (in our case it is the clause R'').

The reason for using the heuristic above is as follows. In our implementation of the BPE procedure, variables x_i are processed in the order they are numbered. Let clause R' be chosen to eliminate $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$. Suppose that after elimination of all symmetric x_5-boundary and $\overline{x}_5$-boundary points and removing the clauses with variable x_5 from G (see Figure 2), R' is the only clause of G falsified by $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$. Then, say, $\boldsymbol{p_0}$ is an x_{10}-boundary point of G. This means that when processing variable x_{10}, the point $\boldsymbol{p_0}$ and the point *flip*($\boldsymbol{p_0}$,x_{10}) need to be eliminated (and hence may cause an extra run of the SAT-solver used for finding $l(x_{10})$-boundary points.) On the other hand, if R'' is chosen to eliminate $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$, no extra work is needed until variable x_{20} is processed. In other words, when eliminating $l(x_i)$-boundary points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$, *gen_resolvent* tries to minimize the number of $l(x_j)$-boundary points the points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ turn into (where $i < j$) after removing the clauses with variable x_i.

Avoiding the generation of boundary points that cannot be eliminated is done by *find_bnd_pnts* as follows. Let $\boldsymbol{q}$ be a partial assignment such that $G_{x_i}(\boldsymbol{q})$=1 and/or $G_{\overline{x}_i}(\boldsymbol{q})$=1. Note that no boundary point $\boldsymbol{p}$ obtained by extension of $\boldsymbol{q}$ can be eliminated by a resolvent of a clause of G_{x_i} with a clause of $G_{\overline{x}_i}$ (because any such a resolvent is satisfied by $\boldsymbol{p}$). To make *find_bnd_pnts* avoid the regions of the search space where G_{x_i} and/or $G_{\overline{x}_i}$ are satisfied, new clauses are added to H by the *dir_clauses* procedure.

Now we explain how avoiding the regions where $G_{x_i} = 1$ is done. (The new clauses meant to avoid regions where $G_{\overline{x}_i} = 1$ are built analogously.) First, a new variable y_c is introduced for each clause of C of G_{x_i}. Then the clauses describing the equality $y_c \equiv C'$ are added to H where C' is C minus x_i. To guarantee that at least one clause of G_{x_i} stays falsified when $x_i = 0$, the clause consisting of the literals $\overline{y_c}$ is added to H.

5 Quantification by Boundary Point Elimination

In this section, we show that elimination of all symmetric x_i-boundary and $\overline{x}_i$-boundary points of a CNF formula G is equivalent to existentially quantifying away variable x_i from G. This means that the BPE procedure simply quantifies away the variables of X_1 (see Figure 2). This result shows that the BPE approach can be used for existential quantification. (And as we argue in Section 6, existential quantification is one of the bottlenecks of CDCL SAT-solvers). At the same time, we believe that the naive algorithm of the BPE procedure, where boundary points are eliminated variable by variable, can be significantly improved.

Proposition 3. *Let $G(x_1, \ldots, x_n)$ be a CNF formula. Let G_{x_i} (respectively $G_{\overline{x}_i}$) be the set of clauses of G having literal x_i (respectively literal $\overline{x}_i$). Let G' be a set of resolvents where, for every resolvent, one parent clause is from G_{x_i} and the other is from $G_{\overline{x}_i}$. Let $G \wedge G'$ do not have any pair of symmetric x_i-boundary and $\overline{x}_i$-boundary points. Then*
$G(\ldots, x_i = 0, \ldots) \vee G(\ldots, x_i = 1, \ldots)$=$(G \setminus (G_{x_i} \cup G_{\overline{x}_i})) \cup G'$.

Proof. See Appendix A.

Proposition 3 implies that the BPE procedure is complete. Namely, by eliminating the symmetric x_i-boundary and $\overline{x}_i$-boundary points for all variables of G, the BPE procedure existentially quantifies away all variables of G. This results either in producing an empty set of clauses (which means that G is satisfiable) or in generating an empty clause (G is unsatisfiable).

6 CDCL SAT-solvers in terms of BPE

In this section, we give a brief analysis of CDCL SAT-solvers in BPE terms. On the one hand, a CDCL SAT-solver is a particular implementation of BPE-SAT. (This is very interesting taking into account that BPE-SAT is a theoretical construction based on a few general observations.) On the other hand, the BPE approach exposes two problems of CDCL SAT-solvers that, we believe, can be addressed in more advanced implementations of BPE-SAT. These problems are important because their solution will enable structure-aware SAT-solvers.

A CDCL SAT-solver is an implementation of BPE-SAT. A key part of a CDCL SAT-solver is its conflict clause generation procedure [10, 13]. Let $F'(X_1, X_2)$ be the set of clauses responsible for a conflict. Here X_1 is the set of variables in which clauses of F' have literals of both polarities and $X_2 = \mathit{Vars}(F') \setminus X_1$. A conflict clause generation procedure is, essentially, the DP procedure [6] that resolves out the variables of X_1 from F' one by one eventually producing a conflict clause C (where $\mathit{Vars}(C) = X_2$). So, a CDCL SAT-solver is very similar to an implementation of BPE-SAT where subformulas F' are subsets of clauses responsible for conflicts. However, the similarity goes much deeper. It can be shown that a conflict clause generation procedure is essentially a special case of the BPE-procedure that produces resolvents eliminating boundary points of subformula F'. Interestingly, all the required boundary points can be trivially obtained from the partial assignment leading to the conflict specified by F'. (The details are given is Appendix B.) So, a CDCL SAT-solver is an implementation of BPE-SAT.

Eager backtracking interferes with finding subformulas F' sharing boundary points with F. Let $F'(X_1, X_2)$ be the set of clauses responsible for a conflict produced by a CDCL SAT-solver in CNF formula F. Let A and B be clauses of F' resolved on variable $x_i \in X_1$ when generating the conflict clause. Let $\boldsymbol{q}$ be an assignment to $\mathit{Vars}(F')$ such that $\boldsymbol{q}$ and $\boldsymbol{q}^*$=$\mathit{flip}(\boldsymbol{q}$,$x_i)$ are symmetric x_i-boundary and $\overline{x}_i$-boundary points of F'. (Although it is not crucial for our reasoning here, the existence of $\boldsymbol{q}$ can actually be proved similarly to Appendix B.)

Importantly, no guarantees can be made that an $l(x_i)$-boundary of the *entire* current formula F is eliminated by adding the resolvent of A and B. For example, it may be the case that no matter how $\boldsymbol{q}$ is extended by assignments to the variables of $\mathit{Vars}(F \setminus F')$, the extended assignment falsifies a clause of $F \setminus F'$ that does not have variable x_i (and so this extended assignment cannot be an $l(x_i)$-boundary point of F). As we show in Section 9, when a SAT-solver only "accidentally" runs into conflicts that lead to generation of resolutions eliminating boundary points of F, its performance may be very poor. One can address this problem by allowing a SAT-solver to make decision assignments after a conflict occurs. (This may lead to finding another conflict specifying a much better subformula F'.) However, it is not clear if this can be done efficiently: eager backtracking is a cornerstone of CDCL SAT-solvers.

CDCL SAT-solvers and existential quantification. BPE-SAT works with a more general class of subformulas F' (whose variables are existentially quantified away) than CDCL SAT-solvers. A natural question is if this is beneficial for BPE-SAT. According to [9], given a resolution proof R that a CNF formula F is unsatisfiable, a non-deterministic CDCL SAT-solver can build another resolution proof that may be larger than R only by $|\mathit{Vars}(F)|^4$. Since BPE-SAT and CDCL SAT-solvers are based on resolution it is tempting to conclude that a *deterministic* CDCL SAT-solver can efficiently simulate a proof generated by an implementation of BPE-SAT. However, this conclusion may turn out to be wrong because the termination conditions of deterministic and non-deterministic CDCL SAT-solvers are different when performing existential quantification.

Suppose that a non-deterministic CDCL SAT-solver performs existential quantification of $F'(X_1, X_2)$ by generating clauses depending on X_2 and proving that they are implied by F'. Such a SAT-solver stops as soon as a set of clauses $F''(X_2)$ forming a solution is generated. On the contrary, a *deterministic* CDCL SAT-solver, even if $F''(X_2)$ is generated has to show that any assignment to X_2 satisfying F'' can be extended (by assignments to variables of X_1) to an assignment satisfying F'. (I.e. such a SAT-solver has to *prove* that F'' is indeed a solution). This may require some extra work that can be even *exponential*!. The observation above implies that developing efficient methods of existential quantification is very important for creation of new powerful SAT-solvers.

7 Applying BPE-SAT to Solving Narrow Formulas

In this section, we describe an implementation of BPE-SAT meant for solving narrow formulas. We picked this class of formulas for the following reasons. First, narrow formulas occur in practical applications. Second, the choice of good subformulas for narrow formulas is very simple.

Let F be a CNF formula. Let $\mathit{Vars}(F) = \{x_1, \dots, x_n\}$. Denote by $\prec$ the order on $\mathit{Vars}(F)$ specified by variable numbering (i.e. $(x_i \prec x_j) \equiv (i < j)$). Denote by F^i the subset of clauses of F that have variable $x_j, j \leq i$. Denote by F^{i*} the set of clauses $F \setminus F^i$. The value of $|\mathit{Vars}(F^i) \cap \mathit{Vars}(F^{i*})|$ (denoted as $w(F, x_i, \prec)$) is called the width of formula F at variable x_i with respect to order $\prec$. The maximum value among $w(F, x_i, \prec)$,$i = 1, \dots, n$ (denoted as $w(F, \prec)$) is called the width of F with respect to order $\prec$. We assume in this section that the order $\prec$ specifies a minimum (or close to minimum) width with respect to all possible orders on X.

Informally, a CNF formula F has a small width if $w(F, \prec) \ll |\mathit{Vars}(F)|$. To apply BPE-SAT to F we use F^i as a subformula $F'(X_1, X_2)$ of F. (Here $X_1 = \{x_1, \dots, x_i\}$ and $X_2 = \mathit{Vars}(F) \setminus X_1$.) The advantage of such a choice is twofold. First, since F is narrow, if i is small then F' is also small, which makes finding boundary points of F' easy. In our experiments the value of i was set to 100. (Informally, the reason for F' being small is that the number of clauses of F with variable x_i is limited by the value of width. Adding a clause C having variables x_i and x_j to F may increase the width $w(F, x_i, \prec)$ or $w(F, x_j, \prec)$ depending on whether $i < j$ or $i > j$. So one cannot add too many short clauses to F without increasing the value of $w(F, \prec)$.) Second, *any* $l(x_i)$-boundary point of F (where $x_i \in X_1$) is also an $l(x_i)$-boundary point of F'

(because the latter contains *all* clauses of F with x_i). So F' satisfies the criterion of good subformulas of Subsection 4.1.

After quantifying away the variables of X_1 from F', a set of clauses $F''(X_2)$ is generated. The clauses with variables of X_1 are removed from F and the clauses of F'' are added to F. (Note that in the pseudocode of BPE-SAT described in Figure 1 we do not remove clauses with variables of X_1 from F. We can do these for narrow formulas because, initially, F' contains all the clauses of F with variables of X_1.) Then the same procedure is applied to F again. That is F' is the set of clauses of F with a variable $x_j, j \leq 2 * i$. (Here we take into account the fact that variables $x_1, \ldots, x_i$ have been removed from F.) Eventually, either an empty clause is derived (the original formula F is unsatisfiable) or all clauses are removed from F (the original formula F is satisfiable).

8 Some Background

SAT-solving of real-life CNF formulas has been dominated by algorithms based on the DPLL procedure [5] (e.g. [10, 11]). Recently, considerable effort has been given to understanding the reasons for success of CDCL SAT-solvers. In [12] (and the follow-up papers) properties of real-life formulas have been studied. In particular, it was conjectured that such formulas may have very small "backdoors" i.e. sets of variables assigning which dramatically reduces formula complexity. In a number of papers (e.g. [9]), the success of CDCL SAT-solvers has been related to the ability of their non-deterministic counterparts to simulate resolution proofs.

Table 1. Solving formulas describing equivalence checking of identical copies of n-bit multipliers

#bits	(#vars, #cls) $\times 10^3$	Picosat #res. $\times 10^6$	Spec. #res. $\times 10^3$	Proof size ratio
6	(0.5, 1.6)	1.2	1.9	631
7	(0.7, 2.2)	7.3	2.7	2,703
8	(1.0, 3.0)	45	3.6	12,500
9	(1.2,3.8)	249	4.7	52,978

In [8], we showed that a resolution proof can be viewed as a process of boundary point elimination. This result allows one to draw some conclusions about properties of deterministic SAT-solvers based on resolution. In particular, the existence of mandatory resolutions (that require explicitly or implicitly looking for boundary points) sheds some light on why SAT-solvers based on the DPLL procedure dominate among the resolution based algorithms.

9 Experimental Results

The goal of experiments was twofold. First, we wanted to relate poor performance of CDCL SAT-solvers on some formulas with non-trivial structure to their inability to identify (and hence eliminate) boundary points. Second, we tested a very simple implementation of BPE-SAT tailored to narrow formulas and compared it with CDCL SAT-solvers. The experiments were run on a Linux machine with Intel Core 2 Duo CPU with 3.16 GHz clock frequency.

Table 2. Comparison of Picosat proofs with specialized ones

#bits	Picosat		Specialized	
	#res.	mand. res. %	#res.	mand. res. %
2	215	77	77	100
3	2,958	66	409	100
4	32,117	38	957	100
5	231,270	24	1,697	100

Table 1 compares the size of resolution proofs for CNF formulas describing equivalence checking of two identical copies of n-bit multipliers. The first column gives the value of n for the multipliers checked for equivalence. The second column shows the number of variables and clauses (in thousands). The next column gives the number of resolutions in the proofs generated by the SAT-solver Picosat [4] (in millions). (Picosat is a predecessor of Precosat, a winner of the SAT-2009 competition. Since Precosat does not generate proofs, we used Picosat instead.) The size of specialized proofs that take into account the formula structure [8] (in thousands of resolutions) is given in the fourth column. The ratio of the sizes of Picosat proofs and specialized ones is shown in the last column.

Table 3. Solving formulas $F(r, m)$. Time limit is $3{\times}10^5$ seconds. Symbol '$*$' marks the timeouts.

(r, m)	(#vars, #cls) $\times 10^3$	width	Minisat (s.)	Picosat (s.)	BPE (s.)	ratio Pico/BPE
(2,800)	(10,27)	8	0.2	0.5	3.3	0.2
(2,1300)	(17,44)	8	0.7	0.7	6.6	0.1
(2,3000)	(39,102)	8	6.0	3.4	25.3	0.1
(3,300)	(11,30)	16	18.9	2.3	6.4	0.4
(3,1000)	(35,100)	16	536	69.6	30.8	2.2
(3,2000)	(70,200)	16	6,456	162	88.0	1.8
(4,200)	(29,86)	29	2,099	548	58.7	9.3
(4,600)	(88,258)	29	$*$	99,820	235	425
(4,1000)	(147,430)	29	$*$	$*$	955	> 302

The results of Table 1 show that in comparison to specialized proofs, Picosat proofs are very large. We also tried other well known SAT-solvers (that do not generate proofs) like Minisat, Precosat and Glucose. In terms of the number of backtracks their performance was similar to Picosat's.

To explain the difference in proof quality, we computed the value of the SMR metric [8] for the Picosat proofs and the specialized ones. The results are given in Table 2. Given a resolution proof R that a CNF formula F is unsatisfiable, the value of SMR-metric is the Share of Mandatory Resolutions in R i.e. share of resolutions eliminating boundary points not eliminated by previous resolutions. (In [8] this metric was called Share of Boundary Resolutions).

In Table 2 we used the same kind of formulas as in Table 1 but of smaller size (computation of SMR metric is expensive). Table 2 shows that the value of SMR metric for specialized proofs remains 100%. (That is these proofs consist entirely of mandatory resolutions.) On the contrary, the value of SMR metric for Picosat's proofs sharply decreases. This can be attributed to the poor choice of subformulas by CDCL SAT-solvers (see Section 6).

Table 3 describes the experiment with solving narrow formulas. We used a class of unsatisfiable formulas $F(r, m)$, $r > 1$, $m > 0$ specifying equivalence checking of *narrow* circuits [3]. (Equivalence checking was employed here just as a simple way to produce *unsatisfiable* formulas. The formulas we used in experiments and their description can be downloaded from [14]). The two circuits N and N' we checked for equivalence had the same high-level structure specified by parameters r and m. Circuits N and N' mimicked the structure of an m-bit adder with r carry bits. So N and N' consisted of a cascade of m blocks communicating with each other by r wires. N and N' had $m + r - 1$ inputs.

Table 4. Proofs generated by Picosat and BPE-SAT for narrow formulas

(r,m)	(#vars, #cls) $\times 10^3$	Picosat		BPE-SAT		Proof size ratio
		#res. $\times 10^3$	mand. %	#res. $\times 10^3$	mand. %	
(3,10)	(0.4, 1.0)	23.4	18.4	2.6	86.5	9.0
(3,20)	(0.7, 2.0)	96.6	10.6	5.3	90.3	18.2
(3,30)	(1.1,3.0)	123	16.5	8.0	92.1	15.4
(3,40)	(1.4,4.0)	206	13.7	10.7	92.9	19.3
(4,10)	(1.4,4.0)	256	17.2	19.3	60.7	13.3
(4,20)	(2.8,8.2)	856	12.2	35.4	75.5	24.2
(4,30)	(4.3,12.5)	3,415	7.5	51.6	80.6	66.2
(4,40)	(5.7,16.9)	3,036	7.7	67.7	83.5	44.8

For solving formulas $F(r,m)$ we used an implementation of BPE-SAT tailored to narrow formulas. Subformulas F' of $F(r,m)$ were chosen as described in Section 7. For finding boundary points we used the same version of Picosat that was employed in other experiments of this section. In formulas $F(r,m)$ of Tables 3, 4 variables were ordered topologically: from outputs to inputs.

The formulas $F(r,m)$ we used in experiments can be relatively easily solved with BDDs. However, neither BPE-SAT nor CDCL SAT-solvers employ subformula hashing used by BDDs. Importantly, it is not clear how a CDCL SAT-solver can benefit from knowing the structure of these formulas. For example, our implementation of BPE-SAT can successfully solve formulas $F(4,m)$ for a variable ordering for which the formula width is 92. This makes the trivial algorithm of quantifying away the variables of X_1 from subformula $F'(X_1, X_2)$ (where one just enumerates assignments to X_2) inefficient. In Appendix C, we also show experimentally that the DP procedure (that is operationally similar to the BPE procedure of Figure 2) is dramatically slower than BPE-SAT on formulas $F(r,m)$.

In Table 3, we compare BPE-SAT with Minisat (version 2.0) and Picosat on formulas $F(r,m)$, $r \in \{2,3,4\}$. The first column gives the value of parameters r and m. The third column specifies the value $w(F,\prec)$ of the formula width (defined in Section 7). The next three columns give the performance of Minisat, Picosat and BPE-SAT in seconds. The last column shows the ratio of Picosat's and BPE-SAT's runtimes.

The formulas $F(2,m)$ are easily solved by each SAT-solver, with Picosat having the best performance. The formulas $F(3,m)$ of Table 3 are hard for Minisat for $m = 2000$. Picosat is competitive with BPE-SAT and scales up well. This changes for formulas $F(4,m)$. For example, when the formula size increases 3 times (from $F(4,200)$ to $F(4,600)$) the runtime of Picosat increases 182 times (from 548 to 99,820 seconds).

In Table 4, we analyze proofs generated by Picosat and BPE-SAT for narrow formulas. The third and fifth columns give the size of proofs generated by Picosat and BPE-SAT (in thousands of resolutions). The ratio of proof sizes is shown in the last column. The proofs of Picosat are up to 66 times larger than those of BPE-SAT even for small values of m. (For greater values of m, the proofs generated by Picosat were too large to process.) The fourth and sixth columns of Table 4 shed some light on why proof sizes are different: the value of SMR metric for the proofs of Picosat is much lower than for those of BPE-SAT.

10 Conclusions

The fact that the resolution proof system is most likely non-automatizable implies that a resolution-based SAT-solver, in general, needs to know the formula structure to be successful. In this paper, we use the Boundary Point Elimination (BPE) concept to study the problem of building structure-aware SAT-solvers. We show that although the behavior of CDCL SAT-solvers from the viewpoint of BPE is quite reasonable, they have at least two flaws. First, eager backtracking of a CDCL SAT-solver makes it hard to generate resolutions that eliminate boundary points of the formula. Second, a CDCL SAT-solver cannot efficiently perform existential quantification. We introduce a template of resolution SAT-solvers called BPE-SAT meant for addressing these problems and hence for building structure-aware SAT-algorithms.

References

1. M. Alekhnovich and A. Razborov. Resolution is not automatizable unless w[p] is tractable. *SIAM J. Comput.*, 38(4):1347–1363, 2008.
2. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. North-Holland, 2001.
3. L. Berman. Circuit width, register allocation, and ordered binary decision diagrams. *IEEE Trans. on CAD of Integr. Circ. and Syst.*, 10(8):1059–1066, 1991.
4. A. Biere. Picosat essentials. *JSAT*, 4(2-4):75–97, 2008.
5. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.
6. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.
7. R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *KR*, pages 134–145, 1994.
8. E. Goldberg. Boundary points and resolution. In *SAT-09*, pages 147–160, Swansea, Wales, United Kingdom, 2009. Springer-Verlag.
9. K.Pipatsrisawat and A.Darwiche. On the power of clause-learning SAT solvers with restarts. In *Proceedings of CP-2009*, pages 654–668, September 2009.
10. J. Marques-Silva and K. Sakallah. Grasp—a new search algorithm for satisfiability. In *ICCAD-96*, pages 220–227, Washington, DC, USA, 1996.
11. M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient sat solver. In *DAC-01*, pages 530–535, New York, NY, USA, 2001.

12. R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *IJCAI*, pages 1173–1178, 2003.
13. L. Zhang and S. Malik. Conflict driven learning in a quantified boolean satisfiability solver. In *ICCAD*, pages 442–449, 2002.
14. http://eigold.tripod.com/benchmarks/narrow_formulas.tar.gz.

11 Appendix

A Proof of Proposition 3

Proof. Denote by H_l and H_r the left and right parts of the equality to be proved i.e. $G(\ldots, x_i = 0, \ldots) \vee G(\ldots, x_i = 1, \ldots)$ and $(G \setminus (G_{x_i} \cup G_{\overline{x}_i})) \cup G'$ respectively.

1) Proving $H_l = 1 \rightarrow H_r = 1$. Let $\boldsymbol{p}$ be an assignment to the variables of H_l for which $H_l(\boldsymbol{p})$=1. By definition of H_l, there is an assignment to x_i such that by adding it to $\boldsymbol{p}$ one obtains an assignment $\boldsymbol{p^*}$ satisfying G. Every clause of H_r is either a clause of G or a resolvent of clauses of G. So $H_r(\boldsymbol{p^*}) = 1$. Since H_r does not depend on x_i, $H_r(\boldsymbol{p})$=1 too.

2) Proving $H_l = 0 \rightarrow H_r = 0$. Let $\boldsymbol{p}$ be an assignment to the variables of H_l for which $H_l(\boldsymbol{p})$=0. Denote by $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ the assignments obtained from $\boldsymbol{p}$ by adding assignments $x_i = 0$ and $x_i = 1$ respectively. By definition of H_l, $G(\boldsymbol{p_0})$= $G(\boldsymbol{p_1})$=0. Let us consider the following two cases.

a) Points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ falsify a clause C of G that does not have variable x_i. Then C is in H_r and so $H_r(\boldsymbol{p})$=0.
b) Points $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ falsify only clauses of G with variable x_i. This means that $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ falsify at least one clause of G_{x_i} and $G_{\overline{x}_i}$ respectively. Then $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$ are symmetric x_i-boundary and $\overline{x}_i$-boundary points. By definition, $G \wedge G'$ does not have such a pair of points. So G' contains a resolvent on x_i that is falsified by $\boldsymbol{p_0}$ and $\boldsymbol{p_1}$. Since this resolvent is in H_r, $H_r(\boldsymbol{p}) = 0$.

B CDCL SAT-solvers are a special case of BPE-SAT

In this section, we show that the conflict clause generation procedure [13] of a CDCL-SAT-solver is a special case of the BPE procedure and hence a CDCL SAT-solver is a special case of BPE-SAT. We assume that the reader is familiar with the basic notions of CDCL SAT-solvers [10, 11]. First, we describe a simple example and then give a formal description.

Example 2. Let F be a CNF formula containing clauses $C_1 = x_1 \vee x_2$, $C_2 = \overline{x}_2 \vee x_3$, $C_3 = \overline{x}_2 \vee x_4$, $C_4 = x_1 \vee \overline{x}_3 \vee \overline{x}_4$ (and some other clauses). Suppose that the decision assignment $x_1 = 0$ has been made in F and the Boolean Constraint Propagation (BCP) procedure is invoked after that. Then the assignment $x_2 = 1$ has to be made to satisfy C_1. This leads to derivation of assignments $x_3 = 1$ and $x_4 = 1$ required to satisfy C_2 and C_3 respectively. At this point, a conflict occurs because clause C_4 becomes unsatisfiable. Denote by F' the set of clauses $\{C_1, C_2, C_3, C_4\}$ i.e. F' is the subformula of F responsible for the conflict.

The conflict clause is derived from F' as follows [13]. The set *Vars*(F') can be partitioned into subsets $X_1 = \{x_2, x_3, x_4\}$ and $X_2 = \{x_1\}$. The set X_1 consists of the variables on which clauses of F' are resolved when producing the conflict clause. Variables of X_1 are used in the order opposite to the one in which these variables were assigned. First, clause C_4 is resolved with C_3 on variable x_4 producing resolvent $R_1 = \overline{x}_2 \vee x_1 \vee \overline{x}_3$. R_1 is resolved with C_2 on variable x_3 producing the resolvent $R_2 = \overline{x}_2 \vee x_1$. Clauses R_2 and C_1 are resolved on variable x_2 producing the conflict clause x_1 (consisting only of variables of X_2).

Now we show that the same conflict clause is produced by the BPE procedure when existentially quantifying away the variables of X_1 from F'. The necessary boundary points can be trivially produced from the partial assignment $\boldsymbol{q}$ of variables of F leading to the conflict. That is $\boldsymbol{q}$ is the assignments to the variables of *Vars*(F') and so $\boldsymbol{q}$ is equal to $(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 1)$. By construction, $\boldsymbol{q}$ falsifies the unsatisfiable clause $C_4 = x_1 \vee \overline{x}_3 \vee \overline{x}_4$. This means, in particular, that $\boldsymbol{q}$ is an $\overline{x}_4$-boundary point of F'. The point $\boldsymbol{q}^*$=*flip*$(\boldsymbol{q}$,$x_4)$ falsifies only clause $C_3 = \overline{x}_2 \vee x_4$ of F'. So $\boldsymbol{q}^*$ is an x_4-boundary point of F'. Adding the resolvent $R_1 = \overline{x}_2 \vee x_1 \vee \overline{x}_3$ of C_4 and C_3 to F' eliminates $\boldsymbol{q}$ and $\boldsymbol{q}^*$ as $l(x_4)$-boundary points. Adding R_1 also quantifies away the variable x_4 of F' (no more clauses of F' can be resolved on x_4). Then C_3 and C_4 are eliminated from F' (as containing variable x_4) and the assignment $x_4 = 1$ is removed from $\boldsymbol{q}$.

Now $\boldsymbol{q}$= $(x_1 = 0, x_2 = 1, x_3 = 1)$ falsifies only the resolvent R_1. So $\boldsymbol{q}$ is a $\overline{x}_3$-boundary point for F'. The point $\boldsymbol{q}^*$=*flip*$(\boldsymbol{q}$,$x_3)$ falsifies only clause $C_2 = \overline{x}_2 \vee x_3$ of F'. So $\boldsymbol{q}^*$ is an x_3-boundary point of F'. Adding the resolvent $R_2 = \overline{x}_2 \vee x_1$ of R_1 and C_2 to F' eliminates $\boldsymbol{q}$ and $\boldsymbol{q}^*$ as $l(x_3)$-boundary points. Adding R_2 to F' also quantifies away the variable x_3 from F'. The clauses R_1 and C_2 are removed from F' and the assignment $x_3 = 1$ is removed from $\boldsymbol{q}$.

Points $\boldsymbol{q}$= $(x_1 = 0, x_2 = 1)$ and $\boldsymbol{q}^*$=*flip*$(\boldsymbol{q}$,$x_2)$ falsify only the resolvent R_2 and clause $C_1 = x_1 \vee x_2$ respectively. Adding the resolvent x_1 of R_2 and C_1 on x_2 to F' eliminates $\boldsymbol{q}$ and $\boldsymbol{q}^*$ as $l(x_2)$-boundary points. Adding x_1 also quantifies away the variable x_2 from F'. Then R_2 and C_1 are removed from F' making the resolvent x_1 the only clause left.

Formal description. Now we give a formal description. (Note that numbering of clauses, variables and resolvents here is different from the example above.) Let F be a CNF formula and $F' = \{C_1, \ldots, C_{k+1}\}$ be the set of clauses responsible for a conflict found by the BCP procedure.. We assume that the set F' is irredundant i.e. every clause C_i,$1 \leq i \leq k+1$ contributed to the conflict. To simplify the notation we make the following two assumptions. First, BCP derived assignments from $C_1, \ldots, C_k$ in the numbering order (i.e. first from C_1, then from C_2 and so on). C_{k+1} is the falsified clause (the cause of the conflict). Second, we assume that clause C_i, $i = 1, \ldots, k$ contains the positive literal of x_i and the assignment $x_i = 1$ was derived from C_i during BCP.

A conflict clause is obtained by resolving clauses of F' in the reverse order. That is, first C_{k+1} is resolved with C_k on variable x_k. Denote their resolvent as R_k. Then R_k is resolved with C_{k-1} on variable x_{k-1} producing resolvent R_{k-1}. Eventually, resolvent R_2 is resolved with C_1 on variable x_1 producing resolvent R_1 which is a conflict clause. (Note that R_i, $1 \leq i \leq k$, has to contain literal $\overline{x}_i$. Otherwise, no clause $C_j, j > i$ contains literal $\overline{x}_i$, which implies that the assignment $x_i = 1$ derived from C_i has not contributed to the conflict. This contradicts our assumption that F' consists only of clauses responsible for the conflict. Clause C_{k+1} has to contain $\overline{x}_k$ for the same reason.) The set $X = \mathit{vars}(F')$ can be partitioned into the set $X_1 = \{x_1, \ldots, x_k\}$ and $X_2 = X \setminus X_1$. That is, X_1 consists of the variables on which clauses of F' were resolved. The conflict clause R_1 consists of the variables of X_2.

Let us show that the same conflict clause R_1 is produced by the BPE procedure by quantifying away the variables of X_1 from F'. Let $\boldsymbol{q}$ be the assignments to the variables of X made before the conflict occurred. The main idea is to show that the BPE procedure uses the same resolutions (as the conflict clause generation procedure above) to eliminate boundary points obtained from $\boldsymbol{q}$.

By construction, $\boldsymbol{q}$ falsifies only clause C_{k+1} of F'. (Clause C_i, $1 \leq i \leq k$ is satisfied by the assignment $x_i = 1$ derived from C_i during BCP). Then $\boldsymbol{q}$ is an $\overline{x}_k$-boundary point of F'. (Clause C_{k+1} contains $\overline{x}_k$). The point $\boldsymbol{q}^{*}$=$\mathit{flip}(\boldsymbol{q}, x_k)$ falsifies only the clause C_k (by construction it is the only clause that has literal x_k). So $\boldsymbol{q}^{*}$ is an x_k-boundary point of F'. Adding to F' the resolvent R_k of C_{k+1} and C_k on x_k eliminates $\overline{x}_k$-boundary point $\boldsymbol{q}$ and x_k-boundary point $\boldsymbol{q}^{*}$. This concludes processing variable x_k of F' (only clauses C_{k+1} and C_k can be resolved on x_k).

After removing C_{k+1} and C_k from F' (as the clauses depending on variable x_k) and removing the assignment to x_k from $\boldsymbol{q}$, one reproduces the same situation as above with respect to variable x_{k-1}. Now $\boldsymbol{q}$ falsifies only the clause R_k of F'. Since R_k contains $\overline{x}_{k-1}$, $\boldsymbol{q}$ is a $\overline{x}_{k-1}$-boundary point of F'. The point $\boldsymbol{q}^{*}$=$\mathit{flip}(\boldsymbol{q}, x_{k-1})$ falsifies only the clause C_{k-1} of F' and so it is an x_{k-1}-boundary point of F'. Adding to F' the resolvent R_{k-1} of C_{k-1} and R_k on x_{k-1} eliminates $\boldsymbol{q}$ as a $\overline{x}_{k-1}$-boundary point and $\boldsymbol{q}^{*}$ as an x_{k-1}-boundary point. This concludes processing variable x_{k-1}. After the BPE procedure is done with all the variables of X_1, F' reduces to resolvent R_1 that is exactly the clause generated by the conflict clause generation procedure described above.

C BPE-SAT and DP procedure

In this subsection, we use formulas $F(r, m)$ described in Section 9 to compare the implementation of BPE-SAT meant for narrow formulas (see Section 7) and the DP procedure [6].

The reason is twofold. First, the DP procedure and the BPE procedure of BPE-SAT are very similar. The only difference is that when quantifying away variable x_i, the DP procedure adds all the resolvents on variable x_i while the BPE procedure adds only resolvents (on variable x_i) that eliminate symmetric x_i-boundary and $\overline{x}_i$-boundary points. So it is interesting to see how beneficial this reduction in the number of generated resolvents is. Second, the DP procedure is exponential only in formula width [7]. So one may think that the good performance of BPE-SAT on formulas $F(r,m)$ for small values of r trivially follows from the theory of [7].

Some experimental results are given in Table 5. (Our implementation of the DP procedure was reasonably efficient. In particular, no resolvent was generated if it was subsumed by an existing clause of the formula.) For each of the three formulas of Table 5 we used two variable orderings (from inputs to outputs and vice versa) that had different width. The first two columns show the value of parameters (r,m) and the number of variables and clauses for each of the three formulas. The values of width are shown in the third column. The next two columns give the runtime of the DP procedure and BPE-SAT (in seconds). The last column shows the ratio of runtimes.

Table 5. DP procedure and BPE-SAT

(r,m)	(vars, cls.) $\times$1000	width	DP (s.)	BPE-SAT (s.)	ratio DP / BPE-SAT[1]
(2,1000)	(13,34)	8	10.3	4.5	2.3
		11	122	4.0	30.5
(3,200)	(7,20)	16	22.1	6.4	3.5
		30	14,150	6.0	2,358
(4,80)	(12,34)	29	14,408	21.6	667
		92	$> 3 \times 10^5$	90.2	$\approx 3.7 \times 10^6$

One can make the following two conclusions based on the results of Table 5. First, the DP procedure is not competitive with CDCL SAT-solvers on formulas $F(r,m)$ even for small values of r. For example, it took about 4 hours for the DP procedure to complete $F(4,80)$ (for the most favorable variable ordering) while Minisat and Picosat solved it in 9.8 and 13.2 seconds respectively. Second, although both DP procedure and BPE-SAT are sensitive to the formula width, the sensitivity of the DP procedure is dramatically higher. In particular, in 300,000 seconds it was able to process only 11 variables out of 12 thousands for the formula $F(4,80)$ for the variable ordering with the width of 92.

[1] To compute the approximate ratio for the formula $F(4,80)$ (with ordering width 92) we used the percentage of variables resolved out by the time the DP procedure was aborted (0.09%).