# Determinization of resolution by an algorithm operating on complete assignments

**Cadence Berkeley Labs**
**1995 University Ave.,Suite 460, Berkeley, California,94704**
**phone: (510)-647-2825, fax: (510)-486-0205**

**CDNL-TR-2007-0106**
**January 2007**

**Eugene Goldberg (Cadence Berkeley Labs), egold@cadence.com**

**Abstract.** "Determinization" of resolution is usually done by employing a DPLL-like procedure that operates on *partial* assignments. We introduce a resolution-based SAT-solver that operates on *complete* assignments and give a theoretical justification for determinizing resolution in such a way. This justification is based on the notion of a *point image* of a resolution proof. We give experimental results confirming the viability of our approach to resolution determinization.

## 1. Introduction

The resolution proof system [1] has achieved outstanding popularity in practical applications. Since resolution is a non-deterministic proof system, any SAT-solver based on resolution, one way or another, has to perform its "determinization". In the state-of-the-art SAT-solvers this determinization is based on using the DPLL procedure [2] that operates on *partial* assignments. The current partial assignment is extended until a clause is falsified. Then, the DPLL procedure backtracks to the last decision assignment and flips it. The search performed by the DPLL procedure can be simulated by so-called tree-like resolution (a special type of general resolution).

The reason for using partial rather than complete assignments is that by rejecting a partial assignment the DPLL procedure may "simultaneously" reject an exponential number of complete assignments. The premise of such an approach is that to prove that a CNF formula $F$ is unsatisfiable one has to show that $F$ evaluates to 0 for all complete assignments.

In this report, we introduce the notion of a point image of a resolution proof that questions the premise above. Let $R$ be resolution proof that a CNF formula $F$ is unsatisfiable. Let $T$ be a set of points that has the following property. For any resolvent $C$ of $R$ obtained from parent clauses $C'$ and $C''$, there are two points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ of $T$ such that

1. $C'(\boldsymbol{p}') = 0$ and $C''(\boldsymbol{p}'') = 0$,
2. $C(\boldsymbol{p}')=C(\boldsymbol{p}'')=0$.

Then the set $T$ is called a point image of $R$.

Given a resolution proof $R$, one can always build its point image whose size is at most twice the size of $R$ (measured in the number of resolution operations). Besides, given a set of points $T$ and a CNF formula $F$, one can always test if $T$ is a point image of a resolution proof by a simple procedure described in this report. This result implies that a resolution proof that a CNF formula $F$ is unsatisfiable can be "guided" by testing the value of $F$ at a sequence of points. Moreover, if $F$ has a short resolution proof of unsatisfiability, the number of "guiding" points is negligible with respect to the size of the entire search space.

Interestingly, a resolution proof $R$ has, in general, an exponential number of point images but not all point images are "equivalent" from the point of view of determinization. Informally, this means that given two different point images $T_1$ and $T_2$ of a proof $R$, one may be able to generate, say, $T_1$ by a much simpler program than $T_2$.

In this report, we introduce a DPLL-like SAT-solver called *FI* (which stands for *F*ind *I*mage) that is inspired by the observation above. Namely, *FI* operates on *complete* assignments. Algorithmically, *FI* can be interpreted as a DPLL like SAT-solver that is allowed to make assignments only to variables of clauses that are falsified by a complete assignment. This complete assignment dynamically changes. However, as we show in this report much more fruitful interpretation of *FI* is to consider it as an algorithm "directly" operating on complete assignments. In this interpretation all the variables are assigned and *FI* only "fixes" some assignments. We give experimental results showing that while *FI* is competitive in the quality of proofs with the state-of-the-art, it has interesting properties that SAT-solvers operating on partial assignments do not have.

This report is structured as follows. In Section 2 we recall basic definitions. In Section 3 we give some justification of our

algorithm and introduce the notion of a point image of a resolution proof. Section 4 describes *FI*. In Section 5, we explain the advantages of the "point" interpretation of *FI* over the traditional interpretation in terms of partial assignments. Section 6 explains why decision making of *FI* works well. In Section 7 we show that the set of points "visited" by *FI* is a point image of the proof *FI* builds. Section 8 compares *FI* with local search and DPLL-based procedures. Experimental results are given in Section 9. We make some conclusions in Section 10.

## 2. SAT and resolution proofs

In this section, we give some basic definitions.

Let $F$ be a CNF formula (i.e. conjunction of disjunctions of literals) over a set $X$ of Boolean variables. The satisfiability problem (SAT) is to find a complete assignment $\boldsymbol{p}$ (called a **satisfying assignment**) to the variables of $X$ such that $F(\boldsymbol{p}) = 1$ or to prove that such an assignment does not exist. If $F$ has a satisfying assignment, $F$ is called **satisfiable**. Otherwise, $F$ is **unsatisfiable**. A disjunction of literals is further referred to as a **clause**. A complete assignment to variables of $X$ will be also called a **point** of the Boolean space $B^{|X|}$ where $B=\{0,1\}$. A point $\boldsymbol{p}$ **satisfies** clause $C$ if $C(\boldsymbol{p})=1$. If $C(\boldsymbol{p})=0$, $\boldsymbol{p}$ is said to **falsify** $C$. Denote by $\boldsymbol{Vars(C)}$ and $\boldsymbol{Vars(F)}$ the set of variables of $C$ and $F$ respectively.

Now we recall basic definitions of the resolution proof system [1]. Let $C_1$ and $C_2$ be two clauses that have opposite literals of a variable $x_i$. Then the clause consisting of all the literals of $C_1$,$C_2$ except those of $x_i$ is called the **resolvent** of $C_1$,$C_2$. (For example if $C_1=x_1 \vee x_3 \vee x_5$, $C_2= x_2 \vee \sim x_3 \vee x_7$ , the resolvent of $C_1$ and $C_2$ is the clause $x_1 \vee x_5 \vee x_2 \vee x_7$ .) The resolvent of $C_1$,$C_2$ is said to be obtained by the **resolution operation**.

The resolvent of $C_1$,$C_2$ is implied by $C_1 \wedge C_2$. So, if an empty clause is derived from clauses of $F$, then $F$ implies an empty clause and so $F$ is unsatisfiable. Hence, the resolution system is sound. It is also complete, that is, given an unsatisfiable CNF formula $F$, one can always generate a sequence of resolution operations resulting in producing an empty clause. This sequence of operations is called a **resolution proof**. The resolution proof system is very important from a practical of view because the best SAT-solvers for solving "industrial" formulas (like Grasp [18], SATO [19], Chaff [13], BerkMin [7], Minisat [3] and Siege) are based on resolution.

## 3. Justification of our approach

In this section, we give a theoretical justification of our approach.

### 3.1 Point image of resolution proof

Let $R$ be a resolution proof that a CNF formula $F$ is unsatisfiable. Let $T$ be a set of points that has the following property. For any resolvent $C$ of $R$ obtained from parent clauses $C'$and $C''$ there are two points $\boldsymbol{p'}$and $\boldsymbol{p''}$of $T$ such that

1 $C'(\boldsymbol{p'}) = 0$ and $C''(\boldsymbol{p''}) = 0$.

2 $C(\boldsymbol{p'})=C(\boldsymbol{p''})=0$.

Then the set $T$ is called a **point image of resolution proof** $\boldsymbol{R}$. The points $\boldsymbol{p'}$ and $\boldsymbol{p''}$ are called a **point image of the resolution operation** over clauses $C'$and $C''$.

The definition of point image of resolution operation above is more general than the one we gave in [5]. Here, instead of the requirement of [5] that $\boldsymbol{p'}$an $\boldsymbol{p''}$ are at Hamming distance 1, we only require that the resolvent $C$ is falsified by $\boldsymbol{p'}$and $\boldsymbol{p''}$. The difference between the two definitions is in assignments to "free" variables of $\boldsymbol{p'}$ an $\boldsymbol{p''}$, that is the variables of the set $Vars(F) \setminus (Vars(C') \cup Vars(C''))$. According to the previous definition, $\boldsymbol{p'}$an $\boldsymbol{p''}$ have to have *identical* assignments to the free variables. The new definition allows one to assign the free variables of $\boldsymbol{p'}$ an $\boldsymbol{p''}$ *arbitrarily*. The advantage of the new definition in the context of test generation is explained in [6]. (It allows one to extract from resolution proofs high quality test sets called *tight* sufficient test sets.) In this report, the new definition allows us to show that the set of points visited by the SAT-solver *FI* (described later) is a point image of the proof *FI* builds.

### 3.2 Building a point image of a resolution proof

Given a resolution proof $R$ that $F$ is unsatisfiable, a trivial way of building a point image $T$ of $R$ is as follows. We start with an empty set $T$. Then for every resolution operation from $R$ over clauses $C'$and $C''$we add to $T$ two points $\boldsymbol{p'}$and $\boldsymbol{p''}$ forming its point image (unless $\boldsymbol{p'}$and/or $\boldsymbol{p''}$ have been added to $T$ before.) Clearly, the size of a set $T$ built this way is at most twice the number of resolution operations in $R$.

### 3.3 Checking if a set of points is an image of a proof

Given a set of points $T$ and a CNF formula $F$, one can test if $T$ is a point image of a resolution proof by the following procedure. Let $S$ be a set of clauses that initially consists of the clauses of $F$. At every step of this procedure we pick a pair of clauses of $C'$and $C''$of $S$ such that a point image of $C'$and $C''$ is in $T$ and add the resolvent to $S$ unless it is subsumed by a clause of $S$. This procedure has three termination conditions. 1) If a point of $T$ satisfies $F$, then clearly $F$ is satisfiable and $T$ is not a point image. 2) No new clause can be added to $S$ at a step of the procedure. This means that $T$ is not a point image of any resolution proof and so one cannot say whether $F$ is satisfiable or not yet. In other words, the set $T$ is not "large enough" to be a point image of a proof. 3) An empty clause is derived at a step of the procedure. This means that $T$ is a point image of a resolution proof that $F$ is unsatisfiable.

In [6], we show experimentally that the procedure of checking if $T$ is a point image we just described is inefficient. That is, although this procedure dramatically reduces the set of resolution operations that are "allowed", it still performs a large number of "junk" resolutions. That is the majority of generated resolvents do not contribute into the derivation of an empty clause. (As we show in [6], one can make this procedure efficient by adding some extra information.) However, this inefficiency does not matter. We describe this procedure just to show that a small set of points can "encrypt" a proof and that this fact can be established by a deterministic procedure (although inefficient).

## 3.4 A proof has a huge number of point images

Let *Res* be the resolution operation of a proof $R$ (that a CNF formula $F$ is unsatisfiable) over clauses $C'$ and $C''$. The operation *Res*, in general, has a huge number of point images because the values of points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ forming a point image of *Res* are specified only for the variables of $C'$ and $C''$. For the variables of $F$ that are not in $C'$ and $C''$, points $\boldsymbol{p}'$ and $\boldsymbol{p}''$ may have arbitrary values.

Since a point image of a resolution proof $R$ is essentially the union of point images of resolution operations comprising $R$, the latter has a huge number of point images. However, not all point images of $R$ are equivalent in the sense that some images are more "regular" and so can be more easily built by a deterministic algorithm. So, resolution, being a non-deterministic proof system, does not distinguish between different point images of $R$. On the other hand, the fact that different point images of $R$ have different complexity of building them, implies that an algorithm operating on complete assignments can be used to "determinize" resolution. The idea of such an algorithm it to try to build a "regular" point image of a resolution proof that $F$ is unsatisfiable.

# 4. Description of *FI*

The procedure of subsection 3.3 shows that one can use complete assignments to "guide" a resolution proof. The size of a "guiding" set $T$ (if it is "irredundant") is at most twice the size of the proof the set $T$ "guides". In this section, we introduce a resolution-based SAT-solver called ***FI*** (*F*ind *I*mage) that operates on complete assignments (points). Although *FI* is inspired by the ideas of Section 3 it does not look for a point image of a resolution proof "directly". Instead, we formulate *FI* as a DPLL-like procedure whose decision making is driven by a complete assignment.

To simplify understanding of *FI* operation we first give its description in terms of partial assignments (subsection 4.1). The interpretation of *FI* in terms of complete assignments is given in subsection 4.2. (The merits of this interpretation are listed in Section 5.) In the following subsections of this section we describe *FI* in more detail.

## 4.1 Description in terms of partial assignments

The pseudocode of *FI* (without restarts) is shown in Figure 1. In this subsection, we describe *FI* from the viewpoint of a traditional SAT-solver i.e. in terms of partial assignments. (We will denote this interpretation of *FI* as *FI**.) *FI** can be viewed as a regular SAT-solver that uses a complete assignment $\boldsymbol{p}$ as an "oracle" in decision making. This complete assignment is initially generated in line 2 of Figure 1. Then the set $M(\boldsymbol{p})$ of clauses falsified by $\boldsymbol{p}$ is computed (line 3). After that, *FI** follows the well-known procedure [18] used by the current state-of-the-art resolution based SAT-solvers. The only difference is that *FI** maintains two additional entities: a complete assignment $\boldsymbol{p}$ and the set $M(\boldsymbol{p})$ of clauses falsified by $\boldsymbol{p}$. When *FI** makes an assignment to a variable $x_i$ either during BCP (line 5) or decision making (line 12) it checks if the chosen value of $x_i$ is equal to the value of $x_i$ in $\boldsymbol{p}$. If these values are equal, then no recomputation of $\boldsymbol{p}$ or $M(\boldsymbol{p})$ occurs. Otherwise, a new complete assignment $\boldsymbol{p}'$ is produced from $\boldsymbol{p}$ by flipping the value of $x_i$ and the set $M(\boldsymbol{p}')$ is computed (by recomputation of $M(\boldsymbol{p})$). If an unsatisfiable clause is found (line 5), *FI** backtracks without changing the current point $\boldsymbol{p}$.

```
1  FI*(F)
2  {p=generate_initial_point(F);
3   M=find_falsified_clauses(F, p);
4   while (true)
5     {if (BCP(F,M,p) == conflict)
6        {level = analyze(F);
7          if (level == 0)   return(UNSAT);
8         else  backtrack(F,p,M);
9          }
10       else // no conflict yet
11         if (M =∅) return (SAT);
12         else make_assignment(F, p, M);
13     }
14  }
```

**Figure 1. Pseudocode of *FI****

The current complete assignment $\boldsymbol{p}$ and the set $M(\boldsymbol{p})$ are used in *FI** solely for the purpose of decision making. Namely, the next variable to be assigned is picked *only* among variables of clauses of $M(\boldsymbol{p})$ i.e. of clauses that are currently falsified by $\boldsymbol{p}$.

## 4.2 Description in terms of complete assignments

Now we give an interpretation of the procedure of Figure 1, in terms of complete assignments. (We will refer to this interpretation as *FI* without a star symbol.) In this interpretation no free (i.e. unassigned) variables exist. Instead of assigning a value to a free variable $x_i$ (as it is done by *FI** in line 12) , *FI* just *fixes* the *existing* assignment to $x_i$ in a current point $\boldsymbol{p}$. Thus, in all the points $\boldsymbol{p}$ visited after variable $x_i$ has been fixed, the value of $x_i$ stays the same. On the other hand, undoing an assignment to variable $x_i$ in *FI** corresponds in *FI* to "unfixing" the assignment to $x_i$ in the current point. It means that now the value of $x_i$ can change in generated points $\boldsymbol{p}$ until the next time the value of $x_i$ is fixed. A conflict in *FI* is the situation when $M(\boldsymbol{p})$ contains a clause $C$ of the current formula such that all the assignments setting the literals of $C$ to 0 have been fixed. (So no point satisfying $C$ can be obtained from $\boldsymbol{p}$ without unfixing one of the assignments that has been fixed before.)

## 4.3 Decision making of *FI*

The decision making of *FI* is based on variable activity as it was proposed in [13] and later used in many other SAT solvers. The activity of variables is computed similar to BerkMin [7]. But in contrast to BerkMin we compute the activity of literals (as it was done in Chaff) rather than variables. Let $lit(x_i)$ be a literal of variable $x_i$. If $lit(x_i)$ occurs in $k$ clauses of the current formula that were involved in the last conflict, then the activity of $lit(x_i)$ is incremented by $k$. After a fixed number of conflicts, the activity of literals is divided by a small constant as it was first done in Chaff [13].

We experimented with two different procedures of decision making that performed well. The first procedure is BerkMin-like. If $M(\boldsymbol{p})$ contains conflict clauses, then *FI* picks the conflict clause $C$ of $M(\boldsymbol{p})$ that was derived most recently. (Note that since $M(\boldsymbol{p})$

contains clauses falsified by *one* point, no variable can have literals of both polarities in clauses of $M(\boldsymbol{p})$). *FI* picks the most active literal $lit(x_i)$ among variables of $C$ and fixes the assignment to $x_i$ setting $lit(x_i)$ to 1. This assignment does not necessarily satisfy $C$. Suppose $C$ contains the literal $\sim lit(x_i)$ (i.e. the opposite of $lit(x_i)$) which means that $\boldsymbol{p}$ has the assignment to $x_i$ that sets $lit(x_i)$ to 1. Then *FI* fixes the current assignment to $x_i$ and so $\boldsymbol{p}$ and the set $M(\boldsymbol{p})$ stay unchanged. Otherwise, (i.e. if $C$ contains $lit(x_i)$) *FI* flips the current value of $x_i$ thus satisfying $C$ and only then fixes the new value of $x_i$. This entails changing $\boldsymbol{p}$ and recomputing $M(\boldsymbol{p})$. If $M(\boldsymbol{p})$ does not contain a conflict clause, then *FI* fixes the assignment setting to 1 the most active literal $lit(x_i)$ among the variables of clauses of $M(\boldsymbol{p})$.

The second decision-making procedure of *FI* fixes the assignment setting to 1 the most active literal $lit(x_i)$, where $x_i$ is a variable of a clause of $M(\boldsymbol{p})$. (As we explained above this assignment does not necessarily satisfy a clause of $M(\boldsymbol{p})$.) That is the second procedure works exactly as the first decision-making procedure when $M(\boldsymbol{p})$ does not contain conflict clauses.

## 4.4 BCP procedure and conflict clause analysis

The BCP procedure of *FI* is identical to that of a generic DPLL based SAT-solver. So the BCP procedure of *FI* is "*global*" in contrast to its decision making. As it was described above, in its decision making, *FI* fixes assignments *only* of variables that occur in clauses of $M(\boldsymbol{p})$. In the BCP procedure, *FI* keeps track of all the unit clauses regardless of whether they are falsified or satisfied by $\boldsymbol{p}$ and so regardless of their presence in $M(\boldsymbol{p})$ . (In terms of fixed assignments, a clause is *unit*, if all its literals but one are set to 0 by fixed assignments.)

*FI* also employs a traditional conflict analysis and conflict clause generation whose description can be found in [13].

## 4.5 Restarts

*FI* uses occasional restarts as it was suggested in [8]. (I.e. once in a while, *FI* abandons the current search tree to start a new one.) After a restart, *FI* inherits the last complete assignment $\boldsymbol{p}$ obtained before abandoning the previous search tree. As we will show in Section 9, this allows *FI* to benefit from making frequent restarts.

## 4.6 Initial point generation

Currently *FI* uses the following procedure for generation of an initial point $\boldsymbol{p}$. This procedure makes a "decision" assignment to a variable $x_i$ of the formula and runs BCP procedure to make all the implied assignments. If implied assignments to a variable contradict each other, one of them is picked randomly. This goes on until all variables are assigned. This procedure may vary in how variable $x_i$ and its assignment are chosen. Variable $x_i$ can be chosen randomly or according to a particular order. An assignment to $x_i$ can be also picked randomly or according to some heuristic.

# 5. Which interpretation is "better"?

In this section, we justify the interpretation of *FI* as an algorithm operating on complete assignments (we will refer to it as a **point interpretation** of *FI*). We explain why the point interpretation is much more "fruitful" than the interpretation in terms of partial assignments described in subsection 4.1. For the sake of simplicity, in the discussion of this section, we assume that *FI* does not make restarts and so builds a single search tree.

## 5.1 DPLL still needs determinization

In this subsection, we try to draw the line between the notions of a heuristic and determinization. By determinization, we mean a reduction of the proof space in such a way that some "reasonably good" choices still remain. In a sense, DPLL can be viewed as determinization of general resolution because it reduces the type of allowed proofs but the proof space still contains "good" solutions. By a heuristic, we mean a way to find a particular proof out of a space of proofs (already reduced). For example, making an assignment satisfying the largest number of unsatisfied clauses is a heuristic. More formally, one can view a heuristic of a DPLL-like procedure as a function that, given a CNF formula and a set of already assigned variables, computes the next assignment to make.

We believe that it is wrong to consider the DPLL procedure as "almost" an algorithm that just needs a heuristic to make DPLL deterministic. In reality, DPLL is still a "highly non-deterministic" proof system that needs determinization. A strong indicator of that is the fact that tree-like resolution is most likely non-automatizable [15]. Non-automatizability implies that given a class of CNF formulas with short proofs in tree-like resolution, there is no efficient algorithm for finding those proofs. This essentially means that effective and efficiently computable heuristics are impossible for DPLL. (This does not contradict the fact that very good decision-making heuristics have appeared recently. These heuristics, in general, cannot help find a small proof if, say, the original CNF formula has a small unsatisfiable core. See subsection 9.3.)

One way to solve the problem is to reduce non-determinism of the DPLL procedure even more and then search for heuristics. Otherwise, heuristics may be very fragile and unrobust and it will be hard to say whether a particular heuristic "makes sense" or is just tailored for a small finite set of benchmarks. It will also be hard to predict whether this heuristic will work for a slightly larger set of benchmarks.

Decision making based on using complete assignments provides much "deeper" determinization of DPLL than that of current resolution based SAT-solvers. "Extra" determinization is achieved by reducing the choices of decision-making only to the variables of clauses falsified by the current point. As we will see in Section 6 such reduction indeed can be viewed as a determinization of DPLL, because the reduced set of variables still contains "good" choices.

## 5.2 Interpretation in terms of complete assignments is much more "fruitful"

Probably, the most important reason why we strongly suggest using the point interpretation of *FI* is as follows. It is much more fruitful to consider a DPLL-like procedure in terms of point interpretation of *FI* than the other way around. In terms of partial assignments, *FI* is a DPLL procedure with "odd" decision making . This decision making looks extremely strange because, in terms of partial assignments, a SAT-solver tries to cover the entire search space as fast as possible. However, *FI* makes

decisions that are based on a complete assignment i.e. on a miniscule part of the space of all assignments.

On the other hand, the point interpretation of *FI* allows one to study and classify existing decision making heuristics and generate new ones. First of all, note that DPLL can simulate *FI* while *the opposite is not true*. Indeed, any choice of an assignment to fix made by *FI* can be simulated by DPLL as an assignment to a "free" variable. However, a search tree built the DPLL procedure, in general, can not be simulated by *FI*. The reason is twofold.

Firstly, the set of variables to fix is limited to those of clauses that are falsified by the current point $\boldsymbol{p}$. Secondly, $\boldsymbol{p}$ can not be changed arbitrarily. Recall, that *FI* either fixes an assignment to a variable $x_i$ that agrees with $\boldsymbol{p}$ (in which case $\boldsymbol{p}$ does not change) or it flips the value of $x_i$ and then fixes it. In this case, the new point is at distance 1 from the previous one. In other words, *FI* can not flip more than one bit of $\boldsymbol{p}$ at a time and moreover, only bits corresponding to the variables of clauses falsified by $\boldsymbol{p}$ may be flipped. This means that only a *negligible* number of search trees built by DPLL can be simulated by *FI*. Surprisingly, in spite of this dramatic reduction of the proof space, *FI* is competitive in the number of backtracks with current SAT-solvers operating on partial assignments (see Section 9.) Our explanation of this fact is given in Section 6.

To simulate the DPLL procedure, *FI* should be "extended". By testing these extensions one can study various decision heuristics. Basically there are two ways to extend *FI*. First extension is to allow *FI* to make "non-local" moves (where more than one bit of $\boldsymbol{p}$ can be changed at once) and/or to allow to flip the bits of $\boldsymbol{p}$ that are not in a clause falsified by $\boldsymbol{p}$. Importantly, one can control the degree of "non-locality", for example, by limiting moves that are allowed.

The second type of extension is to maintain more than one complete assignment. This type of extension seems to be more promising for the following reason. By allowing "non-local" changes of $\boldsymbol{p}$ one makes *FI* more "non-deterministic" and hence more DPLL-like. On the other hand, by keeping more than one point, *FI* can improve its decision making without introducing "extra" non-determinism.

## 5.3 Traditional semantics of DPLL may be very misleading

The underlying semantics of a SAT-solver operating on partial assignments is that this SAT-solver has to cover the entire search space. (At each step it generates branches $x_i=0$ and $x_i=1$, which guarantees complete search space coverage.) Suppose that an initial formula $F$ contains a unit clause $x_i$ or such clause can be "easily" derived from $F$. From the semantics of DPLL it follows that by setting $x_i$ to 1, one reduces the search space by half and so "half the work" is done by making this assignment. However, it is well known that derivation of unit clauses usually does not reduce the amount of work to do as dramatically as one can conclude from the search space covering semantics.

An explanation of this phenomenon can be easily given in terms of point images. Suppose, an original CNF formula $F$ contains a unit clause $x_i$ and $F$ is unsatisfiable. Then in a point image $T$ of a proof $R$ that $F$ is unsatisfiable there will be only *one* point falsifying clause $x_i$. (Here we assume that both $R$ and $T$ are not "artificially" inflated by obvious redundancies.) The rest of the points of $T$ will have $x_i$ assigned to 1. So the fact that $x_i$ is fixed at 1 does not mean that a lot of work is done because only *one point* of the subspace $x_i=0$ is in $T$. (On the other hand, if the unit clause $x_i$ is derived and this derivation is "long", then a lot of points with $x_i=0$ may be added to $T$. This means that the derivation of this clause is a "large piece of work".)

# 6. Comparison of decision making based on complete and partial assignments

In this section, we discuss one important advantage of decision making of *FI* over SAT-solvers operating on partial assignments like Chaff, BerkMin and many others. We also describe one obvious "flaw" of *FI*'s decision making.

Let $F$ be an unsatisfiable CNF formula and $F'$ be an unsatisfiable subset of clauses of $F$. Let $\boldsymbol{p}$ be a complete assignment to variables of $F$ and $M(\boldsymbol{p})$ be the set of clauses of $F$ falsified by $\boldsymbol{p}$. Note, that regardless of the choice of $\boldsymbol{p}$, at least one clause of $F'$ has to be in $M(\boldsymbol{p})$. So in spite of the fact that the decision-making choices of *FI* are very limited, it still has good decisions to make because variables of $F'$ are always available for decision making. (Here we assume that making decisions on variables of $F'$ is a good idea.) So according to definition of subsection 5.1, using a complete assignment to guide decision-making can be viewed as determinization of tree-like resolution.

One may think that the argument above is applicable only if the original formula is unsatisfiable and has a small irredundant core of clauses. In reality, this argument is applicable to *any* CNF formula. Indeed, if an original formula $F$ is satisfiable, after fixing values of some variables, $F$ may become unsatisfiable. Besides, even if the original formula $F$ is irredundant it becomes *redundant* after fixing values of variables. Eventually, unless a satisfying assignment is found, an unsatisfiable clause is encountered. This clause forms an "unsatisfiable core" of $F$ under the current set of assignments.

One more universal source of redundancy is adding derived clauses (e.g. conflict clauses.) On the one hand, if a clause $C$ is implied by $F$, then obviously $C$ is redundant in $F \wedge C$. On the other hand, some clauses of $F$ may become redundant in $F \wedge C$. Adding clauses to $F$ by a resolution based SAT-solver, in general, leads to forming a small unsatisfiable core. Indeed, eventually, an empty clause is derived. This clause forms an unsatisfiable core of the formula $F \wedge D$ where $D$ is the set of derived clauses.

The advantage of *FI*'s decision making mentioned above, due to its universal applicability, can explain good experimental results of *FI* (see Section 9.) Note that the observation that local search can be used for identifying an unsatisfiable core of the *initial* formula was made in [12].

An obvious disadvantage of the decision making of *FI* in comparison to that of SAT-solvers operating on partial assignments is the restriction on variables whose value can be fixed. Note that we do not contradict our claim above about merits of *FI*'s decision-making. If the current formula $F$ contains an unsatisfiable core $F'$, current point $\boldsymbol{p}$ necessarily falsifies a clause $C$ of $F'$. However, variables of $C$ may be not the best variables of $F'$ to fix. Had we chosen another point $\boldsymbol{p}^*$, it would have

falsified some other clause $C^*$ of $F'$ with "better" variables to fix. However, *FI* can not fix a variable of $C^*$ if it is not in a clause falsified by the current point. (As we mentioned in subsection 5.2, one of the natural extensions of *FI* is to maintain more than one complete assignment, which could mitigate the problem above.)

# 7. Does *FI* build a point image of its proof?

Let $F$ be an unsatisfiable formula and $R$ be a resolution proof found by *FI* when solving $F$. Let $T$ be the set of points visited by *FI*. In this section, we show that after a "natural" extension of the set $T$, it becomes a point image of $R$. In subsection 7.1 we describe this natural extension and in subsection 7.2 we show that the extended set $T$ is a point image of $R$.

## 7.1 Extension of set *T* of visited points

The extension above is due to the fact that during BCP *FI* may fix assignments of variables of clauses that are not in $M(\boldsymbol{p})$. Let $C$ be a unit clause (i.e all the literals of $C$ but one are set to 0 by fixed assignments). Suppose, the assignment satisfying $C$ agrees with the current complete assignment $\boldsymbol{p}$. (So $C$ is not $M(\boldsymbol{p})$.) After this assignment is fixed by *FI*, the clause $C$ may get involved in a conflict without being falsified by any point of $T$. However, the necessary condition for $T$ to be a point image of $R$ is that every clause of $R$ is falsified by a point of $T$. So, obviously, $T$ is not an image of $R$.

Suppose we extend $T$ by the set of points that falsify all clauses that became unit during a run of BCP and from which an (implied) assignment was derived. This extension can be built in the following way. If, during BCP a unit clause $C$ is satisfied by fixing an assignment to a variable $x_i$ that agrees with the current point $\boldsymbol{p}$ we add to $T$ the point $\boldsymbol{p}'$ obtained from $\boldsymbol{p}$ by flipping the value of $x_i$. Obviously, $\boldsymbol{p}'$ falsifies $C$. This extension of $T$ is "natural" in the sense that the only reason why *FI* does not visit the points like $\boldsymbol{p}'$ "explicitly" is that they can not be a satisfying assignment.

## 7.2 *FI* builds image of proof it generates

Let $C_{\text{cnfl}}$ be the conflict clause produced in the latest conflict. The conflict clause generation based on the first-UIP scheme (that is used by *FI*) is well described in [20]. According to this scheme $C_{\text{cnfl}}$ is generated by resolving clauses from which an assignment was deduced at the conflict level and the clause that became unsatisfiable. Let $C_{\text{cnfl}}$ be obtained by resolving clauses $C_1,..,C_k$ of the current CNF formula $F$. Here $C_1,..,C_{k-1}$ are clauses from which assignments were deduced during BCP at the conflict level and $C_k$ is the unsatisfiable clause. (Note that during BCP at the conflict level, assignments may have been deduced from clauses other than $C_1,..,C_{k-1}$. But we are interested only in the clauses involved in the conflict.) Assume that $C_1,..,C_{k-1}$ are numbered in the order in which they were processed by BCP (that is if $i < j$, then an assignment was deduced from $C_i$ *before* another assignment was deduced from $C_j$).

In the first-UIP scheme, $C_{\text{cnfl}}$ is obtained from $C_1,..,C_k$ by resolving them in the "reverse order". That is, first, $C_k$ is resolved with $C_{k-1}$. Then $C_{k-2}$ is resolved with the resolvent of $C_k$ and $C_{k-1}$ and so on.

**Proposition**. The extended set of points $T$ is a point image of the proof generated by *FI*.

**Proof.** We need to show that for each $k$-1 resolution operations performed to generate $C_{\text{cnfl}}$, the extended set $T$ contains two points forming an image of this operation. This can be proven by induction in the number of resolutions. Note that by definition of the conflict situation, the last conflict clause $C_k$ has to be falsified by the current point $\boldsymbol{p}_{\text{cnfl}}$.

*Basis*. The fact that $T$ contains point $\boldsymbol{p}_{\text{cnfl}}$ falsifying $C_k$ and that the variables of $Vars(C_k)$ have fixed assignments is the basis of our inductive proof.

Denote by $R_m$ the clause equal to $C_k$ for $m$=1 or the result of resolving the clauses $C_k,C_{k-1},C_{k-m-1}$ for $m > 1$.

*The inductive statement* of our proof is as follows. We assume that $R_m$ is falsified by the point $\boldsymbol{p}_{\text{cnfl}}$ and all the variables of $Vars(R_m)$ have assignments fixed *before* an assignment was deduced from $C_{k-m-1}$.

Using this assumption we will show that

1) the inductive statement holds for the next value of $m$;
2) $T$ contains a point $\boldsymbol{p}_{\mathbf{m}}$ falsifying the clause $C_{k-m}$ such that $\boldsymbol{p}_{\text{cnfl}}$ and $\boldsymbol{p}_{\mathbf{m}}$ form the point image of the resolution operation over $R_m$ and $C_{k-m}$.

***Proof of the first condition.*** The resolvent $R_{m+1}$ is obtained by resolving $R_m$ and $C_{k-m}$. Denote by $Ded\_var(C_{k-m})$ the variable whose value was deduced from $C_{k-m}$ during BCP. Note that before a value was deduced from $Ded\_var(C_{k-m})$, the literals of the other variables of $C_{k-m}$ were set to 0 by *fixed* assignments. So the clause $R_{m+1}$ is falsified by $\boldsymbol{p}_{\text{cnfl}}$. Note that the assignments to the variables of $Vars(C_{k-m}) \setminus \{Ded\_var(C_{k-m})\}$ were fixed *before* deducing a value of $Ded\_var(C_{k-m})$. So all the literals of $R_{m+1}$ were fixed at 0 before derivation of $Ded\_var(C_{k-m})$.

***Proof of the second condition.*** Let $\boldsymbol{p}$ be the point that was the current complete assignment of *FI* at the time an assignment was deduced from $C_{k-m}$. Denote by $\boldsymbol{p}_{\mathbf{m}}$ the point of $T$ defined as follows. If $C_{k-m}$ was falsified by $\boldsymbol{p}$, then $\boldsymbol{p}_{\mathbf{m}} = \boldsymbol{p}$. If $C_{k-m}$ was satisfied by $\boldsymbol{p}$, then $\boldsymbol{p}_{\mathbf{m}}=\boldsymbol{p}'$ where $\boldsymbol{p}'$ is obtained by the extension of $T$ described above (i.e. by flipping the value of variable $Ded\_var(C_{k-m})$). In either case, $C_{k-m}(\boldsymbol{p}_{\mathbf{m}})=0$. Now we need to show that both $\boldsymbol{p}_{\mathbf{m}}$ and $\boldsymbol{p}_{\text{cnfl}}$ falsify the resolvent $R_{m+1}$ of $C_{k-m}$ and $R_m$. We already showed above that $\boldsymbol{p}_{\text{cnfl}}$ falsifies $R_{m+1}$.

Now we show that $\boldsymbol{p}_{\mathbf{m}}$ sets to 0 all the literals of $R_m$ (except maybe the literal of $Ded\_var(C_{k-m})$) and hence, taking into account that $C_{k-m}(\boldsymbol{p}_{\mathbf{m}})=0$, the point $\boldsymbol{p}_{\mathbf{m}}$ falsifies $R_{m+1}$. Let $lit(x_j)$ be a literal of $R_m$ (where $x_j$ is different from $Ded\_var(C_{k-m})$). Note that according to our inductive statement the value of $x_j$ was fixed *before* variable $Ded\_var(C_{k-m-1})$. Since $Ded\_var(C_{k-m})$ and $x_j$ are different, the assignment $x_j$ was also fixed *before* $Ded\_var(C_{k-m})$ was fixed. This assignment to $x_j$ sets $lit(x_j)$ to 0 (otherwise $\boldsymbol{p}_{\text{cnfl}}$ would satisfy $R_m$). So $\boldsymbol{p}_{\mathbf{m}}$ falsifies $R_{m+1}$.

# 8. Relation of *FI* to DPLL and local search

In this section, we compare our approach with local search SAT-algorithms and SAT-solvers based on the DPLL procedure and give some background information.

There have been many tries to combine local search algorithms pioneered in [16],[17] and SAT-solvers based on the DPLL procedure [2]. In [12], in every node of the DPLL procedure, a local search procedure is invoked to identify the next variable to branch on . This approach was also tried in [9] with the following modification. Before running a local search

procedure at a node of the search tree, dependencies between variables of the current formula were computed. In [14] random backtracking was used to improve the scalability of the DPLL procedure. In [10], BCP was used to correct values of a complete assignment $\boldsymbol{p}$. The values of $\boldsymbol{p}$ were re-assigned in a random order, every assignment being followed by BCP. A complete local search algorithm augmented by clause generation was introduced in [4]. Clause generation was used in [4] for escaping local minima.

The only feature that *FI* shares with local search SAT-solvers is its operating on complete assignments. At the same time, a typical local search procedure [11] has at least one of the following three features: 1) it is incomplete; 2) it tries to optimize a "straightforward" cost function (like the number of falsified clauses); 3) making random decisions plays an important role in SAT-solver's performance. On the other hand, *FI* is complete, does not optimize any "straightforward" cost function and random decisions are not of crucial importance. Probably, the best way to position *FI* is to view it as a resolution-based SAT-solver that operates on complete assignments and so makes one more step away from the DPLL procedure. The SAT-solver of [4] is also complete and based on resolution but it introduces new clauses in "a mechanical way" just to escape a local minimum. Our experiments show that the SAT-solver of [4] generates an enormous number of new clauses and so fails to prove the unsatisfiability of even very small CNF formulas.

# 9. Experimental Results

In this section, we give results of some experiments with an implementation of *FI*. The main objective of experiments was to show that *FI* is competitive with state-of-the-art SAT-solvers in the number of backtracks (or, equivalently, in the number of conflicts). So we used a very simple implementation that lacked the techniques commonly employed to speed up a SAT-solver (like fast BCP, efficient formula representation, special treatment of binary clauses and so on.) Besides, we tried to keep our implementation of *FI* as simple as possible (to facilitate changing the code of *FI*). For that reason we do not report runtimes. However in subsection 9.1 we give experimental data suggesting that overhead for maintaining complete assignment $\boldsymbol{p}$ and the set $M(\boldsymbol{p})$ for large CNF formulas is small. We also discuss how this overhead can be further reduced.

**In our experiments, we used the first decision-making heuristic described in subsection 4.3 for the formulas of Tables 1-5. (It was slightly modified for the formulas of Table 4 as described below). For the formulas of**

Table 6 we used the second decision-making heuristic of subsection 4.3 In all the experiments, (except for those reported in Table 5) a restart was performed every 150 conflicts.

In subsection 9.2, we compare *FI* with other SAT-solvers in terms of the number of conflicts. In the following three subsections we try to highlight some advantages of decision making of *FI*. In subsection 9.3, we show that *FI* is able to find unsatisfiable subformulas that can not be found by SAT-solvers operating on partial assignments. Subsection 9.4 shows that employing complete assignments allows *FI* to use more frequent restarts because subproofs found in different iterations become more "coherent". Finally, in subsection 9.5 we show that decision making of *FI* is more "precise" than that of a SAT-solver operating on complete assignments. As a result, *FI* makes fewer decisions.

## 9.1 About efficient implementation of *FI*

In contrast to regular resolution-based SAT-solvers, *FI* has to maintain the set $M(\boldsymbol{p})$ of clauses of $F$ falsified by the current complete assignment $\boldsymbol{p}$, which may affect *FI*'s performance. Every time a decision or implied assignment to a variable is fixed and it *disagrees* with the current complete assignment $\boldsymbol{p}$, the latter changes and $M(\boldsymbol{p})$ has to be recomputed. Let $x_i$ be the flipped variable (whose value is fixed after flipping) and $\boldsymbol{p}'$ be the point obtained from $\boldsymbol{p}$ by flipping the value of $x_i$. The recomputation involves removing clauses that are satisfied by $\boldsymbol{p}'$ from $M(\boldsymbol{p})$ and adding to $M(\boldsymbol{p})$ the clauses falsified by $\boldsymbol{p}'$ (that were satisfied by $\boldsymbol{p}$). Note that only clauses having literals of $x_i$ are involved in recomputation of $M(\boldsymbol{p})$.

Removing from $M(\boldsymbol{p})$ the clauses satisfied by $\boldsymbol{p}'$ is "cheep". If for every literal of $F$, one maintains the subset of clauses of $M(\boldsymbol{p})$ that have this literal, then one just needs to "empty" this subset for the corresponding literal of $x_i$. Finding the clauses with variable $x_i$ that one has to add to $M(\boldsymbol{p})$ is more time consuming. In a naive implementation, one needs to examine every clause $C$ of $F$ with the literal of $x_i$ of the corresponding polarity and check if $C$ is falsified by $\boldsymbol{p}'$. To reduce the complexity of this part of updating $M(\boldsymbol{p})$ one can use watched literals introduced in [13][19].

The idea is as follows. For every clause $C$ of $F$ we pick a literal $lit(x_j)$ of $C$ such that the variable $x_j$ is not fixed and the value of $x_j$ in the current complete assignment sets $lit(x_j)$ to 1. So $C$ is satisfied by the current complete assignment and hence $C$ is not in $M(\boldsymbol{p})$. (This watched literal is different from the two watched literals used to check if $C$ is unit). Then, to add the clauses of $F$ that are falsified after flipping the value of $x_i$, one just needs to examine those of them for which the corresponding literal of $x_i$ is "watched". So, $C$ will be accessed only if its watched literal is set to 0 by a fixed value. Then a new watched literal is searched for in $C$. If there is no variable of $Vars(C)$ that is not fixed and whose assignment satisfies $C$, then $C$ is added to $M(\boldsymbol{p})$. Otherwise, a new watched literal is picked.

In regular BCP, to check if a clause is unit, two watched literals are maintained and this clause is accessed every time *at least one* of them is assigned. The check if a clause $C$ has to be added to $M(\boldsymbol{p})$ as described above is performed only if *one* watched literal is assigned. That is checking if a clause $C$ should be added to $M(\boldsymbol{p})$, is performed less frequently then checking if $C$ is unit.

As we mentioned above, one needs to recompute $M(\boldsymbol{p})$ *only* if the assignment to $x_i$ disagrees with the value of $x_i$ in the current complete assignment. Table 1 shows how often *FI* had to recompute $M(\boldsymbol{p})$. The second, third and fourth columns of Table 1 contain the number of variables, clauses and generated conflicts (in thousands). The fifth column contains the number of assignments (both decision and deduced) in millions. The last column of Table 1, gives the percentage of assignments for which *FI* had to recompute $M(\boldsymbol{p})$.

One can make the following two conclusions from Table 1. First, for all formulas, the number of assignments where *FI* had to recompute $M(\boldsymbol{p})$, was smaller than that of assignments where the value to be fixed agreed with the current complete assignment. Second, while for small size formulas (like c3540, c5315)

recomputations of $M(\boldsymbol{p})$ occurred in more than 1/3 of all assignments, for larger formulas this percentage dropped below 10%. This means that the overhead for maintaining $M(\boldsymbol{p})$ in *FI* for large formulas should be negligible. The reason why the percentage of assignments requiring recomputation of $M(\boldsymbol{p})$ dropped for large formulas was that for those formulas the majority of clauses remained satisfied by the original complete assignment. So only a small fraction of the clauses appeared in $M(\boldsymbol{p})$.

**Table 1. Percentage of times *FI* had to recompute $M(\boldsymbol{p})$**

| Name | #Vars $* 10^3$ | #Clauses $* 10^3$ | #Cnfl. $* 10^3$ | #assgns $* 10^6$ | disagr. (%) |
|---|---|---|---|---|---|
| 2bitadd_10 | 0.6 | 1.4 | 219 | 9.1 | 21.9 |
| c3540 | 3.5 | 9.3 | 101 | 35.3 | 34.0 |
| c5315 | 5.4 | 15.0 | 42 | 11.7 | 38.2 |
| 6pipe | 15.8 | 395 | 112 | 64.2 | 3.4 |
| ci | 218 | 639 | 9.5 | 135 | 5.8 |
| ldv4.0.100 | 308 | 902 | 68 | 1335 | 8.1 |
| raven.50 | 756 | 2,243 | 30 | 349 | 8.0 |
| s104 | 1306 | 3864 | 7.1 | 57 | 3.7 |
| smv | 1377 | 4213 | 12.0 | 552 | 3.8 |

## 9.2 Comparison in terms of the number of conflicts

Although the efficiency of the current implementation of *FI* can be significantly improved, its performance was sufficient to collect statistics on a large variety of formulas. First, we give results of applying *FI* to *Dimacs* formulas (Table 2) and some other known families of formulas (Table 3). *Dimacs*, *Beijing*, *blocksworld*, *bmc* formulas can be downloaded from [21]. Formulas *bmc1* (consisting of subclasses *barrel*, *longmult*, *queueinvar*) are described in [22]. Formulas *vliw-sat.1.0*, *fvp-unsat.1.0* and *Npipe* (of *fvp-unsat.2.0*) can be found in [23].

**Table 2. Dimacs formulas**

| Name | #formulas | *Forklift* #conflicts | *Minisat* #conflicts | *FI* #conflicts |
|---|---|---|---|---|
| aim | 72 | 3,303 | 3,587 | **3,256** |
| bf | 4 | 774 | 383 | **379** |
| dubois | 13 | **3,062** | 4,904 | 3,260 |
| hanoi | 2 | **26,156** | 65,428 | 223,040 |
| hole | 5 | 227,102 | 1,538,350 | **56,884** |
| ii | 41 | 6,505 | 4,088 | **1,254** |
| jnh | 49 | 2,151 | 2,096 | **2,069** |
| par16 | 10 | **42,934** | 47,568 | 70,915 |
| par8 | 10 | 304 | 162 | **83** |
| pret | 8 | 4,342 | 6,892 | **2,942** |
| ssa | 8 | 744 | 367 | **348** |

We compare *FI*'s results (in terms of the number of conflicts) with that of *Forklift* , the winner of the SAT-2003 contest in the industrial category, and *Minisat* , the runner-up of the SAT-2005 contest in the industrial category [3]. In Table 3, if *Minisat* was not able to finish all formulas, we report the number of conflicts only for the formulas it solved and give the number of unsolved formulas (in parentheses). The main conclusion we draw from Table 2 and Table 3 is that decision making of *FI* is quite competitive with those of resolution based SAT-solvers operating on partial assignments.

Since the decision-making of *FI* is extremely "local", such a result is hard to understand from the viewpoint of the current DPLL semantics (that every complete assignment of the search space has to be "covered"). However, this result is implied by the theory of Section 3 showing that resolution proofs can be driven by complete assignments. Besides, in Section 6 we explained why the decision-making of *FI* should work. The results of experiments seem to confirm our arguments.

**Table 3. Some other known formulas**

| Name | # formulas | *Forklift* #conflicts | *Minisat* #conflicts (#aborted) | *FI* #conflicts |
|---|---|---|---|---|
| Beijing | 16 | 494,534 | > 721,258(1) | **106,896** |
| blocksworld | 7 | **2,116** | 4,732 | 8,209 |
| bmc | 13 | 54,098 | **44,195** | 48,568 |
| bmc1 | 31 | **1,033,434** | 1,326,812 | 1,568,729 |
| planning | 6 | 29,415 | **17,153** | 24,426 |
| Velev's formulas | | | | |
| vliw-sat.1.0 | 100 | 679,827 | 1,413,027 | **527,416** |
| fvp-unsat.1.0 | 4 | 101,991 | 180,240 | **92,333** |
| 3pipe | 4 | **24,738** | 66,567 | 33,856 |
| 4pipe | 5 | **125,850** | 538,932 | 154,321 |
| 5pipe | 6 | 268,463 | 1,261,229 | **231,975** |
| 6pipe | 2 | 218,461 | >470,779(1) | **176,067** |
| 7pipe | 2 | 386,396 | > 0 (2) | **211,667** |

## 9.3 Finding unsatisfiable subformulas

In Table 4 we consider the performance of *Forklift*, *Minisat* and *FI* on 8 "artificial" formulas with small unsatisfiable subformulas. Formulas f2k10_*i* and f3k50_*i*, *i*=1,2,3 are obtained from formulas f2k10 and f3k50 by random permutation of variables. The formula f2k10 was obtained by adding to a hard random formula *F* of 2000 variables the clauses of a random unsatisfiable formula *G* of 10 *new* variables. The formula f3k50 was obtained in the same way as f2k10. The only difference is that formula *F* has 3000 variables and *G* has 50 new variables. (Note that we got results similar to those of Table 4 even if variables of *F* and *G* had a "weak" overlap.)

**Table 4. Formulas with small unsatisfiable cores**

| Names | *Forklift* #conflicts | *Minisat* #conflicts | *FI* #conflicts |
|---|---|---|---|
| f2k10_1 | > 2,897,607 | > 997,377 | 756 |
| f2k10_2 | > 2,904,869 | > 997,376 | 935 |
| f2k10_3 | > 2,898,438 | > 997,376 | 908 |
| f2k10 | 4 | 6 | 1,209 |
| f3k50_1 | > 6,402,459 | > 5,050,044 | 48 |
| f3k50_2 | > 6,507,802 | > 5,050.047 | 18,127 |
| f3k50_3 | 44 | 66 | 18,890 |
| f3k50 | 71 | 62 | 15,447 |

As we mentioned in Section 6, one of the advantages of operating on complete assignments is easy identification of unsatisfiable subformulas. For a formula $F \wedge G$ of Table 4, no matter how *FI* picks a complete assignment ***p***, the set $M(\boldsymbol{p})$ will contain a clause of the unsatisfiable subformula *G*. In this experiment, we modified the decision-making heuristic of *FI* as follows. For every clause *C* of the formula, its activity was maintained. This activity was computed as the number of conflicts in which *C* was involved. Every 100 decisions *FI* picked the clause $C'$ of $M(\boldsymbol{p})$ with the "lowest" activity and fixed an assignment to a variable of $C'$. This way we made *FI* look for unsatisfiable subformulas (because the modification above made *FI* fix an assignment to a variable of *G* once in a while). Note that this modification does not change *FI*'s performance on other formulas much because in 99% cases *FI* makes a "regular" decision. As one can see from Table 4, *Forklift* and *Minisat* easily solved three formulas, but for the rest of them, they "got stuck" in the hard subformula *F*. On the other hand, *FI* easily solved all 8 formulas.

## 9.4 More frequent restarts, smaller proofs

As we mentioned in subsection 4.5, after a restart, *FI* uses the last point reached after the previous restart as the initial point ***p*** of the new iteration. This makes subproofs generated by *FI* in different iterations more "coherent". In this subsection, we give experimental evidence of *FI*'s benefiting from making frequent restarts.

In Table 5, we apply *FI* to equivalence checking formulas. We compare *FI* to its CounterParT (called *CPT*(*FI*)) operating on partial assignments. *CPT*(*FI*) gets as close to *FI* as it is possible without maintaining a complete assignment ***p*** and the set of falsified clauses $M(\boldsymbol{p})$. The only difference between the two is that *CPT*(*FI*) makes decisions taking into account *all clauses* that remain unsatisfied by the current set of fixed assignments while *FI* makes decisions based *only* on the clauses of $M(\boldsymbol{p})$. (But literal activity computation, conflict clause generation and so on are identical.)

Either SAT-solver was run in the "rare" restart mode (a restart occurs every 150 conflicts) and the "frequent" restart mode (a restart occurs every 5 conflicts). In the rare restart mode, both SAT-solvers generated similar numbers of conflicts (251,165 for *CPT*(*FI*) and 236,937 for *FI*). On the contrary, in the frequent restart mode, *FI* noticeably reduced the number of conflicts to 138,927 while *CPT*(*FI*) did not (255,024 conflicts).

**Table 5. Equivalence checking formulas**

| Name | *CPT*(*FI*) | | *FI* | |
|---|---|---|---|---|
| | rare restarts | freq. restarts | rare restarts | freq. restarts |
| c1355 | 5,241 | 2,563 | 3,832 | **1,960** |
| c1908_bug | 14,161 | 4,199 | 3,575 | **3,088** |
| c1908 | 22,449 | 16,628 | 16,807 | **4,096** |
| c2670_bug | 117 | 51 | **3** | **3** |
| c2670 | 5342 | 4,668 | **2,572** | 2,629 |
| c3540_bug | **0** | **0** | 743 | 59 |
| c3540 | 65,017 | **46,636** | 96,548 | 46,818 |
| c432 | 659 | **556** | 657 | 632 |
| c499 | 2,766 | 2,090 | 2,762 | **1,296** |
| c5315_bug | 3,483 | 7,885 | 2,705 | **514** |
| c5315 | 47,918 | 42,618 | 41,942 | **22,342** |
| c7552_bug | **175** | 2,174 | 296 | 1,645 |
| c7552 | 80,644 | 120,526 | 60,450 | **50,143** |
| c880-s | 3,193 | 4,430 | 4,045 | **3,702** |
| **Total** | 251,165 | 255,024 | 236,937 | **138,927** |

## 9.5 Making fewer decisions

**In this subsection, we show that *FI*, in general, makes fewer decisions per conflict than a typical resolution based SAT-solver. In**

Table 6, we give some results of applying *FI* and *Minisat* to large BMC formulas. (The names of satisfiable formulas are marked with '*' in the first column). The number of variables and clauses (in thousands) is given in second and third columns. For both *FI* and *Minisat*, we report the number of conflicts and the number of decisions. When solving these formulas, *Minisat* and *FI* generated about the same number of conflicts (243,773 and 226,860 respectively). However, *Minisat* made almost 5 times the number of decisions made by *FI* (15,436,775 and 3,192,567 respectively).

# 10. Conclusions

We introduced a new resolution based SAT-solver operating on complete assignments. As a theoretical justification for the new solver, we showed that for a resolution proof *R*, there is always a set of points *T* completely "specifying" *R*. The size of *T* is at most twice the size of *R* measured in the number of resolution

operations. Experimental results show the viability of our approach. Determinization of resolution by an algorithm operating on complete assignments seems to be a promising way for building more powerful resolution-based SAT-solvers.

**Table 6. Bounded model checking formulas**

| Name | #vars | #clause | *Minisat* | | *FI* | |
|---|---|---|---|---|---|---|
| | $*10^3$ | $*10^3$ | #confl. | #decis. | #confl. | #decis. |
| **CI** | 218 | 639 | 9,930 | 46,943 | 10,885 | **42,723** |
| **PA0** | 331 | 980 | 234 | 14,945 | 24 | **119** |
| **SMV*** | 1,377 | 4,212 | 32,896 | 195,594 | 12,320 | **51,085** |
| **BAR*** | 199 | 591 | 532 | 98,965 | 253 | **773** |
| **BIU16*** | 76 | 228 | 147 | 3,243 | 12 | **96** |
| **EMIF3*** | 1,879 | 5,582 | 8,459 | **86,309** | 40,330 | 119,113 |
| **GMTX*** | 234 | 684 | 70 | 10,248 | 0 | **1** |
| **LBQ*** | 166 | 538 | 53,957 | 10,208,072 | 16,069 | **1,933,380** |
| **LIR*** | 301 | 881 | 4,057 | 371,577 | 1507 | **11,705** |
| **PA1*** | 741 | 2,194 | 13,312 | **406,077** | 19,633 | 517,172 |
| **S104*** | 1,306 | 3,864 | 6,485 | 2,204,831 | 6,536 | **13,947** |
| **W2_2*** | 350 | 1,014 | 65 | 718,935 | 11 | **1,108** |
| **LDV.100** | 308 | 902 | 59,986 | 338,201 | 78,194 | **247,112** |
| **PA.25** | 337 | 995 | 2,328 | 94,515 | 96 | **815** |
| **PA.50** | 726 | 2,145 | 46,014 | 604,583 | 7,901 | **59,218** |
| **RAVEN.25** | 340 | 1,010 | 779 | 4,084 | 546 | **2,156** |
| **RAVEN.50** | 756 | 2,242 | 4,522 | **29,653** | 32,543 | 192,044 |
| Total | | | 243,773 | 15,436,775 | 226,860 | **3,192,567** |

## References


[1] L.Bachmair, H.Ganzinger. Resolution theorem proving in A.Robinson, A.Voronkov editors, The Handbook of Automated Reasoning, chapter 2, vol. 1,19-99. Elsevier Science Pub.2001.

[2] M.Davis, G.Longemann, D.Loveland. *A Machine program for theorem proving*. Communications of the ACM. -1962. -V.5. -P.394-397.

[3] Een N., Sorensson N. *An extensible SAT-solver*. Proceedings of SAT-2003 in LNCS 2919, pp.503-518.

[4] H.Fang, W.Ruml. *Complete Local Search for Propositional Satisfiability.* Proc. of 19th National Conference on Artificial Intelligence, 2004, pp.161-166.

[5] E.Goldberg. *Determinization of resolution by an algorithm operating on complete assignments.* SAT-2006, LNCS 4121, pp.90-95

[6] E.Goldberg. *On bridging simulation and formal verification*. Technical Report CDNL-TR-2006-1225, December 2006. Available at http://eigold.tripod.com/papers/ssim.pdf.

[7] E.Goldberg, Y.Novikov. *BerkMin: a Fast and Robust SAT-Solver.* DATE-2002, Paris,pp. 142-149 .

[8] C.P.Gomes, B. Selman, H.Kautz. *Boosting combinational search through randomization*. Proceedings of International Conference on Principles and Practice of Constraint Programming. - 1997.

[9] D.Habet,C.M.Li,L.Devendeville, and M.Vasquez. *A hybrid approach for SAT*. International Conference on Principles and Practice of Constraint Programming, 2002, pp. 172-184.

[10] E. A. Hirsch, A. Kojevnikov. *UnitWalk: A new SAT solver that uses local search guided by unit clause elimination.* Annals of Mathematics and Artificial Intelligence 43(1-4):91-111, 2005

[11] H.Hoos, T.Stutzle. *Stochastic Local Search: Foundations and Applications.* Morgan Kaufmann, San Francisco (CA), USA, 2004.

[12] B.Mazure, L.Sais, and R.Gregoire. *Boosting complete techniques thanks to local search methods.* Annals of Math. and Artif. Intelligence vol. 22 (1998), pp. 319-331.

[13] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik. *Chaff: Engineering an Efficient SAT Solver*. In: Proceeding of the 38th Design Automation Conference (DAC'01), 2001.

[14] S.Prestwich. *Local search and backtracking vs. non-systematic backtracking.* AAAI Fall Symposium on Using Uncertainty Within Computation. November 2-4, 2001, North Falmouth, Cape Cod, MA,pp.109-115.

[15] M. Alekhnovich, A. Razborov. *Resolution is Not Automatizable Unless W[P] is Tractable*, FOCS-2001, pp.210-219.

[16] B. Selman H. Levesque, D. Mitchell. 1992. *A New Method for Solving Hard Satisfiability Problems*. AAAI-92, pp. 440-446.

[17] B.Selman, H.A.Kautz and B.Cohen. *Noise strategies for improving local search.* AAAI-94, Seattle, pp. 337-343, 1994..

[18] J.P.M.Silva, K.A.Sakallah. *GRASP: A Search Algorithm for Propositional Satisfiability.* IEEE Transactions of Computers. -1999. -V. 48. -P. 506-521.

[19] H.Zhang. *SATO: An efficient propositional prover*. Proceedings of the International Conference on Automated Deduction. -July 1997. -P.272-275.

[20] L. Zhang, C. Madigan, M. Moskewicz, S. Malik. *Efficient Conflict Driven Learning in a Boolean Satisfiability Solver*., Proceedings of ICCAD 2001, San Jose, CA, Nov. 2001.

[21] http://www.intellektik.informatik.tu-darmstadt.de/ SATLIB/benchm.html

[22] http://www.lri.fr/~simon/satex/satex.php3

[23] http://www.ece.cmu.edu/~mvelev/