CAMA: A Multi-Valued Satisfiability Solver

Cong Liu[1] Andreas Kuehlmann[1,2] Matthew W. Moskewicz[1]

[1] University of California at Berkeley, CA, USA
[2] Cadence Berkeley Labs, Berkeley, CA, USA

Abstract

This paper presents the multi-valued SAT solver CAMA. CAMA generalizes the recently developed speed-up techniques used in state-of-the-art binary SAT solvers, such as the two-literal-watching scheme for Boolean constraint propagation (BCP), conflict-based learning with identifying the first unique implication point (UIP), and non-chronological back-tracking. In addition, a novel minimum value set (MVS) technique is introduced for improving the efficiency of conflict-based learning. By analyzing the conflict clauses, MVS can potentially prune conflicting space that has not been searched before. Two different decision heuristics are discussed and evaluated. Finally the performance of CAMA is compared with Chaff using on a one-hot-encoding scheme. The experimental results show that, for MV-SAT problems with large variable domains, CAMA outperforms Chaff.

1 Introduction

Boolean satisfiability (SAT) has many applications in the area of electronic design automation (EDA), artificial intelligence (AI), and operations research (OR). It resembles the core of many problems in the field of computer-aided design (CAD) of integrated circuits, including logic synthesis, functional verification, equivalence checking, timing analysis, and automatic test pattern generation (ATPG) [1]. SAT is a classical NP-complete problem – thus its algorithmic solution is believed to have exponential worst-case complexity. Many approaches have been proposed to efficiently solve practical SAT instances; most of them are based on the Davis-Putnam-Logemann-Loveland (DPLL) branching procedure [2, 3] and local search [4, 5]. Some of the advanced SAT solvers include GRASP [6], SATO [7], POSIT [8], Chaff [9], and BerkMin [10].

Many practical decision tasks in CAD can be formulated as constraint satisfaction problem over a multi-dimensional, multi-valued (MV) solution space. The application of a binary SAT solver necessitates an encoding of the multi-valued dimensions using a set of Boolean variables, e.g. by applying a binary encoding or one-hot-encoding (OHE). In general, such encoding requires the specification of additional constraints, which exclude encoded values that do not occur in the original formulation. For example, if a six-valued domain variable is encoded using three binary variables, the remaining two possible value assignments must be excluded from the solution space by corresponding constraints.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICCAD'03, November 11-13, 2003, San Jose, California, USA.
Copyright 2003 ACM 1-58113-762-1/03/0011 ...$5.00.

In this paper we describe a generalized SAT solver that genuinely works on a multi-valued representation of the original problem. Our approach is based on a formulation that uses a set of clauses combining multi-valued literals. Compared to a binary formulation, it represents the problem more compactly and avoids additional clauses for excluding unused values. Furthermore, the presented bit-parallel processing of Boolean Constraint Propagation (BCP) and conflict-based learning provides an efficient implementation working on the native structure of the problem.

The rest of the paper is organized as follows. Section 2 presents some basis definitions and notations used in this paper. A brief introduction of Davis-Putnam-Logemann-Loveland procedure and decision heuristics are outlined in Section 3 and Section 4, respectively. The generalized two-MV-literal watch scheme is described in Section 5. In Section 6 the conflict analysis is explained in detail. Experimental results are presented in Section 7 and conclusion are given in Section 8.

2 Preliminaries

Definition 1 : $x_i \in X$ denotes a *multi-valued variable* with *domain* $P_i = \{0, 1, \ldots |P_i| - 1\}$.

Definition 2 : An *assignment* to a multi-valued variable restricts the possible values it can assume. A variable x_i is called *assigned* to the *variable value set* v_i if x_i can take any value from $v_i \subseteq P_i$ but no value from $P_i \backslash v_i$. If $|v_i| = 1$, the variable assignment is called *complete*, otherwise it is *incomplete*. If all assignments to the variables $x_i \in X$ are complete, the set is called *full assignment*, otherwise it is referred to as *partial assignment*.

Definition 3 : A *multi-valued literal* $x_i^{s_i}$ is a Boolean function defined by:

$$x_i^{s_i} \equiv (x_i = \gamma_1) + \ldots + (x_i = \gamma_k)$$

where $\gamma_j \in s_i \subseteq P_i, j = 1, 2, \ldots, k$. s_i denotes the *literal value set*.

For example, $x_2^{\{1,4\}}$ evaluates to true if x_2 is assigned to 1 or 4. Obviously $x_i^{\emptyset} \equiv 0, x_i^{P_i} \equiv 1$.

Definition 4 : The *valuation* of a multi-valued literal $x_i^{s_j}$ with respect to a variable assignment of x_i to value set v_i is a multi-valued function defined as follows:

$$V(x_i^{s_j})|_{v_i} = \begin{cases} 1 & \text{if } v_i \cap s_j = v_i \quad (i) \\ 0 & \text{if } v_i \cap s_j = \emptyset \quad (ii) \\ X1 & \text{if } v_i \cap s_j = s_j \quad (iii) \\ X2 & \text{otherwise} \quad (iv) \end{cases}$$

$V = 1$ and $V = 0$ means that the literal evaluates to true and false, respectively. $V = X1$ or $V = X2$ denotes an unknown literal value which can usually be treated indistinctly. However, differentiating between the last two cases provides some valuable insight into the relation of the sets v_i and s_j and will be exploited in Section 5.

Lemma 1 : If a multi-valued variable is completely assigned, then the valuations of all corresponding multi-valued literals are uniquely determined to be either true or false.

Proof: If a multi-valued variable is assigned to a specific value, then all corresponding multi-valued literals either contain this value in their value sets, or not. According to Definition 4, it satisfies either case (i) or case (ii), respectively. □

Definition 5 : A *multi-valued clause* is a logical disjunction of one or more multi-valued literals. A MV-clause is denoted as:

$$\omega_k = \sum x_i^{s_j}$$

For example, $\omega_2 = (x_3^{\{1,3\}} + x_2^{\{1,4,5\}} + x_5^{\{0\}})$.

Lemma 2 : A multi-valued clause evaluates to true, if at least one of the multi-valued literals evaluates to true. It evaluates to false, if all the multi-valued literals evaluates to false.

The proof of Lemma 2 follows directly from Definition 5.

Definition 6 : If one literal of a clause evaluates to $X1$ or $X2$ and the remaining literals evaluate to false, the unassigned literal is called a *unit literal*, and the corresponding clause is *unit*. A *conflict* occurs when all literals in a clause evaluate to false; the clause is then referred to as *conflicting clause*.

Definition 7 : A formula in *multi-valued conjunctive normal form* (MV-CNF) is the logical conjunction of a set of multi-valued clauses.

In this paper, we consider multi-valued SAT problems that are represented by MV-CNF formulas.

Lemma 3 : An MV-CNF formula evaluates to true, if all its multi-valued clauses evaluate to true.

The proof of Lemma 3 follows directly from Definition 7.

Definition 8 : A multi-valued SAT problem is *satisfiable*, if there exists a *full assignment* for which the MV-CNF formula evaluates to true. The corresponding full assignment is referred to as a *solution*. The problem is *unsatisfiable*, if no such solution exists.

Definition 9 : A *decision* in a multi-valued SAT solver is the selection of a variable from the set of unassigned or incompletely assigned variables, and its assignment to a refined value set. This assignment is referred to as *decision assignment*.

Definition 10 : *Multi-valued resolution* combines two MV-clauses $\omega_1 = \sum x_i^{s_j} + x^s$ and $\omega_2 = \sum x'^{s'_j}_i + x^{s'}$ to form a new clause $\omega_{res} = \sum x_i^{s_j} + \sum x'^{s'_j}_i + x^{s \cap s'}$. Variable x and clause ω_{res} are called the *resolving variable* and *resolvent*, respectively.

Lemma 4 : Suppose ω_{res} is a resolvent from clauses ω_1 and ω_2. A full assignment evaluates ω_{res} to true if for this assignment ω_1 and ω_2 evaluate to true, i.e., $\omega_1 \wedge \omega_2 \Rightarrow \omega_{res}$.

Proof: For a full assignment, the value of the resolving variable x is uniquely contained in one of the sets of the partition $\{P \setminus s \cap s', s \cap s'\}$. If the value of x is part of $P \setminus s \cap s'$, then $\omega_1 \wedge \omega_2$ implies that either $\sum x'^{s'_j}_i$ or $\sum x_i^{s_j}$ must evaluate to true. Otherwise, both $x^{s'}$ and x^s evaluate to true. Hence $\omega_1 \wedge \omega_2 \Rightarrow \omega_{res}$. □

Note that the encoding of a multi-valued problem using binary variables results in a less compact CNF formula which has a different structure. For example, the clause $\omega = (x_1^{\{1,3\}} + x_2^{\{1,4,5\}})$ can be one-hot encoded into the binary clause $\omega^b = (x_{11} + x_{13} + x_{21} + x_{24} + x_{25})$ were the x_{ij} denote binary variables that are true iff the MV-variable x_i assumes value j. To make the solution space of the encoded problem consistent with the one of the original MV-problem, a set of additional constraints must be added to the binary formula which exclude all assignments to the x_{ij} that are not one-hot.

For a binary encoding, the values of each MV-variable domain P_i are encoded by a set of n binary variables where $n = log_2|P_i|$. However, the conversion of a MV-clause into a binary equivalent is not straight forward. Each MV-literal potentially requires multiple conjunctions of binary literals; the conjunctions must be resolved for obtaining a CNF clause.

3 Davis-Putnam-Logemann-Loveland Procedure

Solving a MV-SAT problem can be performed by an exhaustive search of the domains of all multi-valued variables, until a full assignment satisfying the MV-formula is found. If such assignment does not exist, the problem is unsatisfiable. One of the most practically successful algorithms solving the satisfiability problem is the Davis-Putnam-Logemann-Loveland (DPLL) procedure for which Figure 1 outlines a typical implementation used in modern solvers (e.g. [6, 9, 10]).

```
Algorithm DPLL () {
   while (decide() == SUCCESS) {
      while (deduce() == CONFLICT) {
         if (analyze_conflict() == FAILURE)
            return UNSAT;
      }
   }
   return SAT;
}
```

Figure 1: General Davis-Putnam-Logemann-Loveland procedure.

When solving an MV-SAT problem, the functionality of the three core procedures must be adjusted to the generalized nature of the multi-valued logic. The procedure *decide* makes a decision assignment. It returns FAILURE, if all variables are completely assigned. Otherwise it returns SUCCESS.

The procedure *deduce* is the deduction process, also referred to as *Boolean constraint propagation* (BCP). It evaluates the MV-clauses based on the assignments made so far. If a conflict is encountered it returns CONFLICT. If a unit clause is identified, the unit literal is forced to be true, i.e., the corresponding variable must take values from the unit literal value set. If there is an earlier assignment to this variable, an intersection of the variable value set and the unit literal value set provides the final *implication assignment*. If no further implications can be derived, *deduce* returns SUCCESS.

The procedure *analyze_conflict* identifies assignments causing the conflict and adds a clause to the formula which represent an

abstraction of unsatisfiable parts of the solution space. The added clause is referred as *conflict-induced clause*, or *learned clause*. The process of constructing the learned clauses is called *conflict-based learning*. A conflict is resolved by *backtracking*. Backtracking undoes all recent assignments up to the most recent decision responsible for the conflict. In many cases not all decisions made so far are responsible for the conflict. In these cases it is possible to backtrack beyond one decision level. This is referred to as *non-chronological backtracking*. If the conflict cannot be resolved, i.e., all decision assignments will cause a conflict, *analyze_conflict* returns FAILURE.

4 Decision Heuristics

In case of MV-variables, a key part of the decision heuristic is the determination of the optimum number of values assigned at a decision. There are two extreme cases: (1) The decision chooses exactly one value from the variable value set, or (2) the decision excludes exactly one value from that set.

The first case, also denoted as *large decision scheme* leads immediately to a completely assigned variable for each decision. As a result, according to Lemma 1, the values of all corresponding literals become either true or false. Therefore, the maximum decision depth is equal to the number of multi-valued variables. The advantage of the large decision scheme is the relatively small decision depths with a rapid reduction of the potential solution space. On the other hand, since the search is restricted to a small subspace, the conflict-induced clauses, derived by the complement of this space, are rather weak and contain little information for constraining the future search.

The second case, also referred to as *small decision scheme*, uses an approach that further refines the current incomplete assignment of the decision variable by removing one value from the variable value set. Obviously, the potential depth of the decision tree increases and can be as large as the sum of the sizes of all variable domains and thus results in a corresponding decision overhead. However, the advantage of this scheme is that since only one value is excluded per decision, the two-value-watch scheme needs to visit only clauses with the excluded value watched. Furthermore, the conflict-induced clauses from small decisions are generally stronger than the ones from large decisions.

The decision heuristic works closely with the deduction process and conflict analysis. In CAMA, the large decision heuristic is applied because it works efficiently in combination with the two-literal-watching scheme and conflict analysis using the MVS technique described in Section 6.2.2. The MVS technique compensates for the potentially weak learning of large decisions by applying MV-resolution which can learn clauses that are stronger than the complement of a conflicting assignment.

The previous discussion about decision heuristics can be extended to implication assignments. If the implied variable is incompletely assigned, we could choose to either process the implication assignment in the deduction engine as normal, or to just update the variable value set and skip the deduction. In Section 7 we evaluate the impact of the two decision schemes based on a set of benchmarks.y

```
W_i: Set of watch literals associated to variable x_i
T_j: Set of non-false literals in clause j

Algorithm two_MV_lit_watch(x_i) {
   foreach l_ij ∈ W_i {
     while (evaluate(l_ij) == FALSE) {
        if (T_j == ∅)
           identify_conflict(j);
           return;
        elseif (|T_j| == 1)
           identify_implication(T_j);
           break;
        else
           W_i = W_i \ {l_ij};
           l_kj = get_next(T_j);
           W_k = W_k ∪ {l_kj};
           break;
}}}
```

Figure 2: Pseudo-code for the two-MV-literal-watching scheme.

5 Boolean Constraint Propagation

For solving practical SAT problems, a significant portion of the run time is spent in BCP. Thus its efficient implementation is critical for the overall solver performance. In CAMA, we generalized the BCP techniques used in current state-of-the-art binary SAT solvers.

5.1 Two-MV-Literal-Watching Scheme

In the two-literal-watching scheme outlined in the algorithm of Figure 2, for every clause two literals are monitored for changes of their corresponding variable assignments. The clause is processed during BCP only when one of the watched literals evaluates to false. Otherwise, only the variable value set needs to be updated which can be done independently of the clauses. When a watched literal evaluates to false, a new watched literal is chosen from the set of non-false literals in the clause. If no such literal is found, the other watched literal is identified as the unit literal, and the clause becomes unit. A conflict occurs if both watched literals evaluate to false.

5.2 Evaluation of Multi-Valued Literals

Figure 3 gives the pseudo-code for the evaluation of a literal with respect to a variable assignment according to Definition 4. We use the C-struct style notations $x.val_set$ and $l.val_set$ to denote the value set of variable x and literal l, respectively. Note that the evaluation is invoked at each step of BCP. Since it is one of the most frequently called functions in the solving process it must be implemented efficiently. By using a bit-parallel representation for the value sets, word-wide computer instructions can be applied to perform the individual evaluation and propagation steps.

There are two other key applications of the literal evaluation. First, a TRUE valuation (case (i) in Definition 4) can be used to identify *redundant implications*, which do not further restrict the values the variable can take. For example, the implication $x_2 = \{1,3,4\}$ is redundant if there was an earlier assignment $x_2 = \{1,3\}$. The evaluation of the unit literal allows the deduction engine to recognize this redundancy and skip the implication.

Second, an evaluation to $X1$ (case (iii) in Definition 4) can be used to identify implication assignments that are not affected by earlier variable assignments. This situation occurs when the final

```
Algorithm evaluate(l) {
    x = get_variable(l);
    imp_val_set = x.val_set ∩ l.val_set;
    if (imp_val_set == x.val_set)
        return TRUE;
    if (imp_val_set == ∅)
        return FALSE;
    if (imp_val_set == l.val_set)
        return X1;
    return X2;
}
```

Figure 3: Algorithm for evaluating a multi-valued literal.

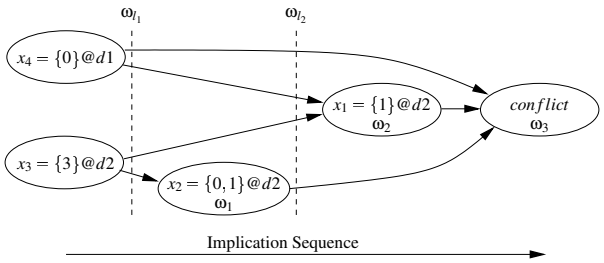


Figure 4: Implication graph and learned clauses for example.

implication assignment is identical to the immediate implication. This information is useful for conflict analysis to determine if this variable should be marked or included in the new clause or not. More details of this process are given in Section 6.2.2.

6 Conflict Analysis

Generally, conflict analysis identifies the set assignments which is responsible for a conflict and uses this set to construct a learned clause. This clause represents a constraint preventing the same assignment to occur in the future by generating implications before the conflict is reached again. Other tasks performed during conflict analysis include: identifying the backtracking level, undoing all recent assignments, and determining the next search direction.

6.1 Binary Conflict Analysis

Traditional conflict analysis utilizes an *implication graph* (IG) which reflects the causal and temporal relationship between decisions and implications. The IG is a directed acyclic graph where the set of vertices represent the assignments and the edges provide the causal relationships between them. The actual clauses causing the implication during BCP are recorded for the assignments. Note that not all assignments made at any point of the search are necessarily involved in a conflict. The goal of conflict-based learning is to derive a compact clause that reflects the conflicting assignments and avoids them in the future.

Learning can be viewed as processing the implications in the IG in reverse order of the original implication sequence and performing resolution operations in order to derive a conflict-induced clause. Each resolution step combines the clause derived so far with the one that caused the implication at that point. Every step of this process corresponds to a cut in the IG that separates the conflict vertex from the ones representing the decisions. An important invariant is that the generated clauses corresponding to the cuts remain under conflict for the current partial assignment, thus, any of them could be learned from the conflict.

GRASP [6] was the first to select a particular cut that facilitates the overall SAT search. By choosing the first cut for which the conflict-induced clause contains only one literal at the current decision level, the clause can be used to "flip" this variable by BCP and thus drive the search forward without the need to explicitly branch to the complemented search space. The corresponding literal vertex in the IG is also referred to as *unique implication point* (UIP).

It should be noted that an efficient implementation of learning in binary SAT solvers, e.g. Chaff [9] does not explicitly perform the resolution steps. Instead, they simply collect the "unresolved" literals at the first cut that has a single literal at the current decision level and combine them to the learned clause. For MV-SAT this simplification would also yield in a clause that drives the search forward, however, in many cases it is weaker than the one derived from exact resolution. In the following we will describe MV-SAT learning using a novel *MVS technique* which produces a stronger learned clause than the traditional "collect and complement" approach.

6.2 Multi-Valued Conflict Analysis

6.2.1 Motivation

We use the following example to compare the learning scheme using the MVS technique with a binary learning scheme based on collecting the "unresolved" literals:

$$\begin{aligned} \omega_1 &= (x_3^{\{1\}} + x_2^{\{0,1\}}) \\ \omega_2 &= (x_4^{\{1,3\}} + x_3^{\{0\}} + x_1^{\{1\}}) \\ \omega_3 &= (x_2^{\{2\}} + x_1^{\{2,3\}} + x_4^{\{3\}}) \end{aligned}$$

Assume that all variables can take values from $\{0,1,2,3\}$ and that the following two decisions were made: $x_4 = \{0\}$ at level 1, and $x_3 = \{3\}$ at level 2. Figure 4 shows the corresponding IG; it is assumed that the implications were made from the left to the right. The traditional learning scheme would collect all unresolved literals and combine their complement to form the following conflict-induced clause:

$$\omega'_{l_1} = (x_4^{\{1,2,3\}} + x_3^{\{0,1,2\}}) \quad (1)$$

This clause is sufficient to imply $x_3 = \{0,1,2\}$ from the assignment $x_4 = 0$ and thus drive the SAT search into unexplored territory. However, the following sequence of exact resolution steps would yield a stronger clause. The first step is:

$$\begin{aligned} \omega_3 &= (x_2^{\{2\}} + x_1^{\{2,3\}} + x_4^{\{3\}}) \\ \omega_2 &= (x_4^{\{1,3\}} + x_3^{\{0\}} + x_1^{\{1\}}) \\ \hline \omega_{l_2} &= (x_2^{\{2\}} + x_4^{\{1,3\}} + x_3^{\{0\}}) \end{aligned}$$

Here we adopt the notation that the antecedents are shown above the line and the consequences below the line. Note that the shown resolution eliminated the literals of x_1. Variable x_1 was

```
A_d:  Assignment stack at decision level d,
Ψ:    Set of marked variables,
Θ:    Set of variables in the learned clause,
V(ω): Set of variables in clause ω,
Σ_d(ω): Set of variables assigned at decision
      level d in clause ω.

Algorithm conflict_analysis(ω_c) {
   d = get_max_dec_level(ω_c);
   if (d == 0) return FAILURE;
   b = d;
   Ψ = Σ_d(ω_c);
   Θ = V(ω_c) \ Σ_d(ω_c);
   foreach assignment α ∈ A_d {
     x_α = get_variable(α);
     if (x_α ∈ Ψ) { /* skip unrelated assignments */
        Ψ = Ψ \ {x_α};
        if (Ψ == ∅) { /* identify UIP* /
           Θ = Θ ∪ {x_α};
           /* new clause from Θ and the x.resolve_set's */
           ω_l = create_learned_clause(Θ);
           b = get_max_dec_level(ω_l);
           imply(ω_l); /* add to database and do BCP */
           break;
        }
        ω_a = get_antecedent(α);
        visit_clause (ω_a, x_α, d, Ψ, Θ);
     }
   }
   back_track(b);
   return SUCCESS;
}
```

Figure 5: Description of the conflict analysis process.

implied by ω_2 with the result that literal $x_1^{\{2,3\}}$ of ω_3 evaluated to false, causing the conflict. By construction, the resolution operation drops x_1 because the literal value sets for x_1 in ω_3 and ω_2 are non-overlapping. The next resolution step yields:

$$\begin{array}{rcl} \omega_{l_2} & = & (x_2^{\{2\}} + x_4^{\{1,3\}} + x_3^{\{0\}}) \\ \omega_1 & = & (x_3^{\{1\}} + x_2^{\{0,1\}}) \\ \hline \omega_{l_1} & = & (x_4^{\{1,3\}} + x_3^{\{0,1\}}) \end{array}$$

Here the literals for x_2 where eliminated. It is easy to show that the resolution steps performed in reverse implication order always lead to an unsatisfied literal value set for the variable implied at the corresponding BCP step. This is because its literal was either (1) part of the conflicting clause and thus its literal evaluated to false for the current assignment, or (2) it evaluated to false for another clause which triggered an implication in a BCP step subsequently leading to the conflict.

Note that clause ω_{l_1} derived by resolution is significantly stronger than clause ω'_{l_1} learned by simply collecting the complemented literals at the learning cut. By adding ω_{l_1} the assignment $x_4 = \{0\}$ implies $x_3 = \{0,1\}$ which in addition to the conflicting assignment $x_3 = \{3\}$ also eliminates the case $x_3 = \{2\}$ from consideration, although this part has never been searched before.

In the following section we describe the minimum value set technique which implements general resolution in an efficient manner.

6.2.2 Minimum Value Set Based Learning

As shown before, general resolution may eliminate values from the value sets of learned clauses which have not been responsible

```
Algorithm visit_clause (ω_a, x_α, d, Ψ, Θ) {
   foreach l_i ∈ ω_a {
      if ((x_i = get_variable(l_i)) != x_α) {
         x_i.resolve_set = x_i.resolve_set ∪ l_i.val_set;
         if (decision_level(x_i) == d)
            Ψ = Ψ ∪ {x_i}; /* still to be processed */
         else /* x_i assigned at earlier decision */
            Θ = Θ ∪ {x_i}; /* to be learned */
      }
      else { /* x_i is the resolving variable */
         if (flag(x_i)) == PREVIOUSLY_ASSIGNED) {
            x_i.resolve_set = x_i.resolve_set ∩ l_i.val_set;
            if (decision_level(x_i) == d)
               Ψ = Ψ ∪ {x_i}; /* still to be processed */
            else /* x_i assigned at earlier decision */
               Θ = Θ ∪ {x_i}; /* to be learned */
         }
      }
   }
}
```

Figure 6: Algorithm for MVS-based learning scheme

for the actual conflict. The *minimum value set* (MVS) technique helps to identify these values such that the value set of each MV-literal in the learned clause is minimal. Similar to the binary case, the conflict-induced clause is constructed by visiting the conflicting clause and the antecedent clauses that led to the conflict. The overall algorithm for MV conflict based learning is shown in Figure 5 and the details of the MVS technique are given in Figure 6.

The conflict analysis begins with analyzing the conflicting clause. The maximal decision level of the variables in the clause determines the chronological decision level at which the clause was processed during BCP. If the conflict clause has a maximal decision level of 0, the conflict is independent of any decision and thus the algorithm immediately returns FAILURE.

During the main loop of the conflict analysis, we maintain a set of variables Θ which is used to construct the learned clause, and a set of variables Ψ which corresponds to assignments at the current decision level that are responsible for the conflict. The two sets together represent a cut in the IG. Using the antecedent clauses recorded for each implied assignment, the algorithm traverses the IG backward and updates Θ and Ψ until a UIP is found. Ψ is updated by adding all variables from traversed clauses that where assigned at the current decision levels and Θ collects the corresponding variables from earlier decision levels. The first UIP is detected when during the processing the last variable is removed from Ψ.

Key of the MVS technique is the computation of an resolution value set $x.resolve_set$ for each variable x. This set reflects the values to be used in the learned clause. The corresponding computation is shown Figure 6 which is based on the inference rules for a generalized MV-resolution described in Definition 10. For each literal of a clause that is processed during the IG traversal the resolution value set of the corresponding variable is updated. For all literals that did not cause the currently processed assignment, i.e., the ones related to non-resolving variables, the set $x.resolve_set$ is simply updated by adding the values from the literal value set. Depending on the decision level at which the variable was assigned, it is added to Θ or Ψ for the current decision level d or an earlier one, respectively.

When processing the resolving variable, it is dropped from the

learned clause unless the implied assignment was affected by a earlier assignment which could have happened on a earlier decision level or on the current one. This is because in contrast to literals of binary variables, MV-literals can be assigned multiple times during BCP before they become true or false. To illustrate this case, consider the following example:

$$\begin{array}{rcl} \omega_1 & = & (x_1^{\{0,2\}} + x_2^{\{1,2,3\}}) \\ \omega_2 & = & (x_3^{\{1,4\}} + x_2^{\{0,1\}}) \\ \hline \omega_l & = & (x_1^{\{0,2\}} + x_2^{\{1\}} + x_3^{\{1,4\}}) \end{array}$$

Suppose there is an earlier level assignment: $x_2 = \{0,3\}$ and a current assignment is $x_1 = \{1\}$. This will lead to the implied assignment $x_2 = \{3\}$ from ω_1. In this example, the implication was affected by the earlier level assignment, which is recorded during BCP by setting a flag PREVIOUSLY_ASSIGNED associated with the recording of the antecedent clause. During conflict analysis, when ω_1 is visited as the antecedent of $x_2 = \{3\}$, the flag indicates that the assignment was affected by an earlier level assignment to its variable. As a result x_2 must be added to Θ to be part of the learned clause. The new resolution value set for x_2 is $\{1\}$ and is constructed by intersecting the literal value set $\{1,2,3\}$ with its current resolution value set $\{0,1\}$, e.g. produced by processing clause ω_2. Note that the resulting resolution value set is smaller than $\{1,2\}$, the complement of the earlier level assignment value set.

An important advantage of the MVS techniques is the side effect that a large decision scheme will not "hurt" the learning efficiency. In other words, a search based on large decisions does not necessarily lead to weak learning. For the example used in Section 6.2.1 the clause ω_{l_1} can also be learned for the decisions $x_4 = \{0,2\}$, $x_3 = \{2,3\}$, which corresponds to a larger search space.

6.3 Original Assignment Storage Mechanism

As stated in Section 3, the final implication assignment is determined by the intersection of the current variable value set and the immediate implication. However, the intersection operation is not reversible, i.e., the original variable value set cannot be restored by performing simple set operations on the final implication and the unit literal value set. As a result, the traditional "undo assignment" mechanism typically used in binary SAT solvers is not applicable.

The idea of the *original assignment storage* (OAS) mechanism is derived from the observation that the exact values of the assignments are never used in the MVS based conflict analysis. Only the variable index of the assignment is needed. One can take advantage of the MVS technique by not storing the "real" assignment value set on the assignment stack, but instead the original value set of the variable just before the intersection operation. The OAS mechanism simplifies the "undo assignment", because all assignments to be restored are on the assignment stack.

7 Experiments

In this section we first describe the set of MV-SAT benchmarks and then report the results of the following four experiments: (1) The two decision heuristics discussed in Section 4 are compared. (2) The efficiency of MVS technique is evaluated by comparing two learning schemes. (3) The performance of CAMA is compared with zChaff [9]. (4) The results of some preliminary experiments on the MV-reasoning for Boolean circuits are reported.

Note that the implementation of CAMA regarding the variable selection heuristic and processing is similar to the architecture of zChaff [9]. This allows a fairly good evaluation of the advantages of the MV scheme in comparison to a binary solver. Other speedup techniques, e.g. the decision processing used in BerkMin [10] are orthogonal and can be added for a more comprehensive solver. For comparing the performance of CAMA with zChaff, we use OHE to transform the MV-SAT problem into a binary SAT problem.

7.1 MVSIS Benchmarks

The MVSIS [11] benchmarks were derived from applications in data mining, artificial intelligence, and other areas. The MV-SAT problems are formulated as an equivalence check of MV-functions before and after minimization using the MVSIS package. Though most of the benchmarks are relatively easy to solve, their statistics provides valuable insight into the decision heuristics and learning schemes. In addition to the standard benchmarks we also created a test suite named *m*4. It includes a series of equivalence checks of MV-functions of the form $f(i,j,k,l) = max(i, min(j, max(k,l))))$ before and after minimization. For the various test instances, we use different variable domain sizes in the original MV-functions, which are indicated by the last digit of benchmark name. The goal is to demonstrate how the solver performs with increasing variable domain size.

7.2 DIMACS Benchmarks

In this test suite, we use five PHOLE instances of the "Pigeon hole" problem from the DIMACS benchmarks in SATLIB [12]. The problem checks whether $n+1$ pigeons can be placed in n holes without two pigeons occupying the same hole. Though the answer is obvious, the "Pigeon hole" problem is known to be one of hardest SAT problems.

An MV-SAT formulation of the problem is straightforward: $n+1$ MV-variables $x_1, \ldots x_{n+1}$ are introduced for the pigeons with variable domains sizes equal to n, the number of holes. An assignment of $x_i = j$ means that pigeon i is placed in hole j. The MV-CNF formula is composed of $n^2(n+1)/2$ MV-clauses representing that pigeon i and j can not be placed in the same hole k which corresponds to the clause $(x_i^{P\setminus\{k\}} + x_j^{P\setminus\{k\}})$.

7.3 MV-Reasoning for Boolean Circuits

In order to evaluate the efficiency of an MV-SAT approach for solving traditional two-valued circuit benchmarks, we converted a set of circuit-based SAT problems into corresponding MV-SAT problems. The conversion is performed by simply clustering multiple binary variables into one MV-variable. This technique can be viewed as a "decoded" approach as opposed to the encoded method applied when using binary solvers for MV-problems.

For the decoding, we first convert the Boolean circuit into an two-input AND/INVERTER graph. We then traverse the AND/INV graph to combine pairs of AND vertices that share one or both fanins into clusters. For each cluster an MV-variable is created that encodes all possible input value combinations. If one or both inputs are shared, a four-value or eight-value variable is

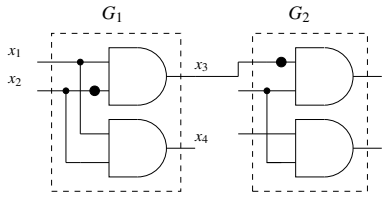


Figure 7: Illustration of connection constraint on clusters.

used, respectively. All gates that are not paired form single-gate clusters.

The MV-clauses of the SAT problem are created based on the connectivity constraints. For example, if an output of cluster G_1 is connected to an input of cluster G_2 two constraints are added to the formula that enforce consistent value assignment for both clusters.

Figure 7 illustrates the clustering approach for four two-input AND gates and two inverters which are marked as dots at the AND inputs. The AND gates are combined into two clusters G_1 and G_2. The possible input values of cluster G_1 are represented by the MV-variable g_1 with domain $\{0,1,2,3\}$ corresponding to $(x_1,x_2) = \{(0,0),(0,1)(1,0),(1,1)\}$. Similarly, cluster G_2 is represented by g_2 for which the value set $\{4,5,6,7\}$ corresponds to $x_3 = 1$. The consistency of the connection from G_1 to G_2 is enforced by the following two clauses:

$$\omega_1 = (g_1^{\{0,1,3\}} + g_2^{\{4,5,6,7\}})$$
$$\omega_2 = (g_1^{\{2\}} + g_2^{\{0,1,2,3\}})$$

In general, clustering can reduce the number of variables and clauses compared to a binary encoding. Thus, it has the potential to reduce the difficulty of the problem.

7.4 Results

All experiments are conducted on a Pentium III 700MHz machine with 256M memory. The reported run times do not include the part spent on reading the CNF formula. All benchmarks are unsatisfiable (UNSAT).

Table 1 gives the result of a comparison of the large and small decision heuristic schemes. In both cases, the MVS learning method was applied. The results suggest that, for MV-SAT problems with large variable domain, the large decision scheme works better than the small one.

The results of comparing two learning schemes are shown in Table 2. The first scheme applies the MVS technique whereas the second scheme uses plain learning of the complemented UIP cut assignment. The advantage of the MVS technique is clearly shown in the reduced number of decisions, learned clauses, and implications needed to solve the instances. Furthermore, the runtime increased dramatically when solving the PHOLE instances without using MVS technique. It indicates that, for hard MV-SAT problems, the MVS technique is critical for the solver performance.

The performance comparison of CAMA to Chaff are presented in Table 3. It is evident that the runtime of both solvers increases exponentially with the variable domain size, however, CAMA significantly outperforms Chaff for problems with larger variable domain.

The preliminary experimental results for using an MV-SAT solver on binary circuit problems are shown in Table 4. Two versions of ISCAS circuits are functionally compared with and without clustering. The results indicate that, for certain applications, solving clustered MV-problems consumes less time. We expect that a more advanced decoding approach that uses larger gate clusters and corresponding variable domains would perform significantly better.

7.5 Discussion

The experiments reveal some fundamental insights into the advantages of using MV-variables as opposed to a one-hot encoding using binary variables. In a OHE approach, additional clauses must be used to explicitly encode the constraint that exactly one binary variable in the OHE of a MV-variable should be true. In contrast, these constraints are implicitly stored, processed, and leveraged in various ways by CAMA during decision, deduction, and learning.

During deduction, a bit-parallel representation is used to store and manipulate the value set of MV-variables and literals. Efficient word-wide operations can significantly reduce unnecessary watch pointer movements. The binary encoded approach dramatically increases the number of literals in each clause, especially for problems with a large variables domain size. Clearly, compared with CAMA, this results in a significantly larger number of watch pointer movements before a unit clause or conflict is identified.

In contrast to the presented MV-method the binary encoded approach accepts only complete assignments and cannot handle incomplete ones. For CAMA, these incomplete assignments lead to more freedom in driving the search in a particular direction. Intuitively, the success of the binary encoded approach depends more on a good decision heuristic when compared with CAMA. This also explains in part the increased number of added clauses in an encoded approach shown in Table 3.

8 Conclusions

In this paper we present the multi-valued satisfiability solver CAMA. It generalizes the efficient SAT search techniques applied in state-of-the-art binary solvers including the two-literal-watching scheme and conflict based learning. A novel MVS technique is described which improves the efficiency of learning by identifying a conflicting subspace that has not been searched before. The presented OAS mechanism facilitates an efficient undo of variable assignments. Our experimental results show that CAMA outperforms Chaff using a one-hot-encoding for solving MV-SAT problems with large variable domain. Furthermore, some preliminary experiments indicate that the decoding of binary SAT problems into MV-SAT problems based on variable clustering could reduce problem size and decrease solution time.

References

[1] P. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1167–1176, 1996.

[2] M. Davis and H. Putnam, "A computing procedure for quantification theory," *Journal of of the Association of for Computing Machinery*, vol. 7, pp. 102–215, 1960.

			Small (Large) Decision Scheme			
Name	Variables	Clauses	Decisions	Added Clauses	Implications	Runtime(s)
pal3	8 (2-3)	77	103 (47)	58 (42)	333 (213)	<0.1 (<0.1)
xor-all	22 (2-4)	183	24 (15)	17 (15)	283 (267)	<0.1 (<0.1)
xor-all-mux	19 (2-8)	186	162 (95)	84 (95)	1296 (1726)	<0.1 (<0.1)
red-adder	12 (2-8)	267	77 (23)	22 (23)	169 (113)	<0.1 (<0.1)
merg-p	12 (2-7)	88	172 (44)	87 (41)	851 (523)	<0.1 (<0.1)
alg5	13 (2-5)	153	414 (123)	244 (121)	2362 (1050)	0.13 (<0.1)
max15	4 (15)	256	1020 (107)	227 (106)	2747 (526)	0.26 (<0.1)
mm4	10 (2-7)	697	895 (409)	638 (406)	4021 (2814)	0.47 (0.25)
mm5	13 (2-9)	2239	3193 (1162)	2276 (1134)	17457 (7184)	4.13 (1.24)
m4_6	9 (2-6)	1073	1057 (363)	736 (361)	3827 (2096)	0.84 (0.35)
m4_8	10 (2-8)	3501	3708 (1151)	2152 (1148)	12165 (2096)	6.25 (2.47)
m4_10	11 (2-10)	8792	11128 (2621)	5076 (2618)	33221 (18246)	39.95 (11.82)
hole6	7(6)	126	669 (129)	630 (129)	5332 (1604)	0.53 (0.10)
hole7	8(7)	196	4500 (321)	4327 (321)	41503 (5073)	13.06 (0.43)

Table 1: Statistics of CAMA solving benchmarks with two different decision heuristics

			Learning without (with) MVS Technique			
Name	Variables	Clauses	Decisions	Added Clauses	Implications	Runtime(s)
pal3	8(2-3)	77	60 (47)	47 (42)	283 (213)	<0.1 (<0.1)
alg5	13(2-5)	284	146 (123)	144 (121)	1265 (1050)	<0.1 (<0.1)
max15	4(15)	256	124 (107)	123 (106)	575 (525)	<0.1 (<0.1)
red-adder	12(2-8)	267	1920 (23)	1920 (23)	7607 (113)	0.59 (<0.1)
mm4	10(2-7)	697	492 (409)	484 (406)	3271 (2814)	0.27 (0.27)
mm5	13(2-9)	2239	1317 (1162)	1286 (1134)	8152 (7184)	1.29 (1.29)
m4_6	9(2-6)	1073	426 (363)	423 (361)	2386 (2165)	0.36 (0.35)
m4_8	10(2-8)	3501	1305 (1151)	1293 (1148)	7631 (6941)	2.50 (2.47)
m4_10	11(2-10)	8792	2992 (2621)	2977 (2618)	19590 (18246)	12.90 (11.82)
m4_12	12(2-12)	18550	6236 (5510)	6236 (5510)	47577 (43239)	36.89 (36.46)
hole6	7(6)	126	719 (129)	719 (129)	4624 (1604)	0.48 (0.10)
hole7	8(7)	196	5039 (321)	5039 (321)	32429 (5073)	16.34 (0.43)
hole8	9(8)	288	49103 (769)	47583 (769)	717041 (15168)	873.67 (2.08)

Table 2: Statistics of CAMA solving benchmarks without and with using MVS technique

Instance	Chaff						CAMA					
	Variables	Clause	Decisions	Added Clauses	Implications	Runtime(s)	Variables	Clause	Decisions	Added Clauses	Implications	Runtime(s)
m4_6	43	1166	881	880	13445	1.00	9(2-6)	1073	363	361	2165	0.35
m4_8	56	3673	2975	2964	52944	7.99	10(2-8)	3501	1151	1148	6941	2.47
m4_10	70	9067	7920	7911	167355	45.37	11(2-10)	8792	2621	2618	18246	11.82
m4_12	84	18952	20686	17715	241786	112.05	12(2-12)	18550	5510	5510	43239	36.46
hole6	42	133	240	130	2331	0.05	7(6)	126	129	129	1604	0.10
hole7	56	204	1962	1814	28708	1.19	8(7)	196	321	321	5073	0.43
hole8	72	297	3450	3198	59504	3.52	9(8)	288	769	769	15168	2.08
hole9	90	415	8870	8539	160729	20.54	10(9)	405	1793	1793	43345	10.28
hole10	110	561	24871	23771	458227	122.01	11(10)	550	5182	5182	146263	67.72

Table 3: Statistics of Chaff and CAMA solving the test suites

			Without (With) clustering			
Name	Variables	Clauses	Decisions	Added Clauses	Implications	Runtime(s)
term1_13_2	173 (103)	662 (478)	228 (212)	148(151)	13989 (11248)	0.22 (0.21)
term1_13_5	159 (89)	602 (434)	71 (59)	53(47)	4329 (2849)	0.1 (0.08)
term1_13_8	184 (116)	666 (498)	129 (157)	95(126)	8058 (6652)	0.17 (0.15)
9symml_13_1	381 (205)	1490 (1068)	270 (279)	253 (268)	59219 (42089)	1.06 (0.95)
C880_13_2	180 (129)	582 (462)	603 (341)	274 (216)	18957(10428)	0.38 (0.25)
C880_13_16	64 (44)	218 (138)	68 (83)	46 (76)	1588 (1532)	0.08 (0.05)
alu4_13_4	1207 (689)	4774 (3506)	888 (663)	746 (547)	270480 (197788)	8.67 (6.04)

Table 4: Statistics of CAMA solving IWLS benchmarks without and with clustering

[3] M. Davis, G. Logeman, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, pp. 394–397, July 1962.

[4] B. Selman, H. A. Kautz, and B. Cohen, "Noise strategies for improving local search," in *Proceedings of the Twelfth National Conference on Artificial Intelligence*, (Menlo Park, CA, USA), pp. 337–343, 1994.

[5] A. M. Frisch and T. J. Peugniez, "Solving non-boolean satisfiability problems with stochastic local search," in *Proceeding of the 17th International Joint Conference on Artificial Intelligence*, 2001.

[6] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

[7] H. Zhang, "SATO: An efficient propositional prover," in *Proceedings of the International Conference on Automated Deduction*, July 1997.

[8] J. W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, May 1995.

[9] M. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S.Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 530–535, June 2001.

[10] E. Goldberg and Y. Novikov, "BerkMin: A fast and robust SAT-solver," in *European Design Automation & Test Conference*, (Paris, France), pp. 142–149, March 2002.

[11] R. K. Brayton and S. P. Khatri, "Multi-valued logic synthesis," in *Proceedings of the International Conference on VLSI Design*, January 1999.

[12] J. Hooker, H. Hoos, and T. Sttzle. DIMACS PHOLE benchmarks available at http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html, 2000.