Integrating Equivalency Reasoning into Davis-Putnam Procedure

Chu Min Li
LaRIA, Univ. de Picardie Jules Verne, 5 rue du moulin Neuf, 80000 Amiens, France
fax: (33) 3 22 82 75 02, e-mail: cli@laria.u-picardie.fr

Abstract

Equivalency clauses (Xors or modulo 2 arithmetics) represent a common structure in the SAT-encoding of many hard real-world problems and constitute a major obstacle to Davis-Putnam (DP) procedure. We propose a special look-ahead technique called equivalency reasoning to overcome the obstacle and report on the performance of an equivalency reasoning enhanced DP procedure on SAT instances containing equivalency clauses derived from problems in parity learning, cryptographic key search and model checking. Our results show that integrating equivalency reasoning renders easy many problems which were beyond DP's reach. We also compare equivalency reasoning with general CSP look-back techniques on equivalency clauses.

Introduction

Consider a propositional formula $\mathcal{F}$ in Conjunctive Normal Form (CNF) on a set of Boolean variables $\{x_1, x_2, ..., x_n\}$. The *satisfiability (SAT) problem* consists in testing whether clauses in $\mathcal{F}$ can all be satisfied by some consistent assignment of truth values (1 or 0) to variables. SAT is fundamental in many fields of computer science, electrical engineering and mathematics. It was the first NP-Complete problem (Cook 1971).

Let l with or without index be a literal. An equivalency clause of length k can be written as

$$l_1 \leftrightarrow l_2 \leftrightarrow ... \leftrightarrow l_k$$

where the operator $\leftrightarrow$ is commutative and associative. The equivalency clause is equivalent to 2^{k-1} CNF clauses. For example, a binary equivalency clause is equivalent to 2 CNF clauses: $l_1 \vee \bar{l}_2$ and $\bar{l}_1 \vee l_2$, and a ternary equivalency clause is equivalent to 4 CNF clauses: $l_1 \vee \bar{l}_2 \vee \bar{l}_3$, $\bar{l}_1 \vee \bar{l}_2 \vee l_3$, $\bar{l}_1 \vee l_2 \vee \bar{l}_3$, and $l_1 \vee l_2 \vee l_3$.

When encoding a hard real-world problem as SAT, one usually structures the problem in layers and makes use of abbreviations and definitions, which often results in a number of equivalency clauses (in their equivalent CNF form) in the final SAT instance. Unfortunately, equivalency clauses of length > 2 generally are hard for Davis-Putnam procedure (DP) (Davis et al. 1962), the best systematic method for SAT.

The reason for the inefficiency of DP procedure on these problems seems to be that equivalency clauses give very few unit clauses throughout the resolution while on other problems DP procedure often deals with many unit clauses under some depth. On the other hand, fixing a variable in equivalency clauses often produces a number of equivalent literals from which an equivalency reasoning can be made to remedy the ineffectiveness of unit propagation. In this paper we show how to integrate equivalency reasoning to solve problems containing both equivalency clauses (called EQ part) and other CNF clauses (called CNF part).

The paper is organized as follows. In section 2 we define equivalency reasoning. In section 3 we present the equivalency reasoning enhanced DP procedure called $EqSatz$. Our approach was originally motivated by the second challenge problem formulated by Selman et al. (1997) at IJCAI'97. Section 4 reports on the performance of $EqSatz$ on the challenge problem. In section 5 we report on and compare the performance of $EqSatz$ with 4 state-of-the-art DPs on various SAT problems involving equivalency clauses. All experimental results are obtained on a Macintosh G3 300 Mhz with 96 Mb memory under Linux system and run time is expressed in seconds. Section 6 discusses related work and Section 7 concludes.

Equivalency Reasoning

An equivalency clause can be negated with the following property:

$$\begin{aligned} \neg(l_1 \leftrightarrow l_2 \leftrightarrow ... \leftrightarrow l_k) &\equiv \bar{l}_1 \leftrightarrow l_2 \leftrightarrow ... \leftrightarrow l_k \\ &\equiv l_1 \leftrightarrow \bar{l}_2 \leftrightarrow ... \leftrightarrow l_k \\ &... \\ &\equiv l_1 \leftrightarrow l_2 \leftrightarrow ... \leftrightarrow \bar{l}_k \quad (1) \end{aligned}$$

Since all equivalency clauses of length > 3 can be simply transformed into ternary equivalency clauses by adding new variables, we only consider binary ($k = 2$) and ternary ($k =$

3) equivalency clauses in this paper. We define six inference rules on them.

$$l_1,\ l_1 \leftrightarrow l_2 \leftrightarrow l_3 \ \vdash\ l_2 \leftrightarrow l_3 \quad (2)$$

$$\bar{l}_1,\ l_1 \leftrightarrow l_2 \leftrightarrow l_3 \ \vdash\ \neg(l_2 \leftrightarrow l_3) \quad (3)$$

$$l_1 \leftrightarrow l_1 \leftrightarrow l_2 \ \vdash\ l_2 \quad (4)$$

$$l_1 \leftrightarrow l_2 \leftrightarrow l_3,\ l_1 \leftrightarrow l_2 \leftrightarrow l_4 \ \vdash\ l_3 \leftrightarrow l_4 \quad (5)$$

$$l_1 \rightarrow (l_3 \leftrightarrow l_4), \bar{l}_1 \rightarrow (l_3 \leftrightarrow l_4) \ \vdash\ l_3 \leftrightarrow l_4 \quad (6)$$

$$l_1 \rightarrow (l_3 \leftrightarrow l_4), \bar{l}_1 \rightarrow (\bar{l}_3 \leftrightarrow l_4) \ \vdash\ l_1 \leftrightarrow l_3 \leftrightarrow l_4 \quad (7)$$

Note that the right side of rule 3 can be rewritten as $\bar{l}_2 \leftrightarrow l_3$ or $l_2 \leftrightarrow \bar{l}_3$ using property 1. All these rules can be realized by a constant number of resolution steps after writing the equivalency clauses in CNF form. We call the application of these rules in a formula $\mathcal{F}$ *equivalency reasoning.*

Binary equivalency clauses play a particular role in our approach. Every time a binary equivalency clause $l_1 \leftrightarrow l_2$ is deduced by rule 2, 3, 5, or 6, l_1 is substituted by l_2 in $\mathcal{F}$. It can be shown that DIMACS[1] dubois* problem is solved in linear time by repeatedly applying rule 5 and the equivalent literal substitution.

Ternary equivalency clauses are represented both in CNF form and by a list of the three involved variables, eventually with a negation operator. For example the equivalency clause $\bar{x} \leftrightarrow \bar{y} \leftrightarrow \bar{z}$ is represented by four CNF clauses and by $\neg(x, y, z)$.

Rule 7 is used to add new ternary equivalency clauses into $\mathcal{F}$. Given three equivalency clauses $x \leftrightarrow u \leftrightarrow v$, $y \leftrightarrow u \leftrightarrow w$, and $z \leftrightarrow v \leftrightarrow w$, one adds a unit clause x, rule 2 applied to the first equivalency clause gives $u \leftrightarrow v$. Then one substitutes u by v in the second equivalency clause before applying rule 5 to obtain $y \leftrightarrow z$. So $x \rightarrow (y \leftrightarrow z)$. Similarly $\bar{x} \rightarrow (\bar{y} \leftrightarrow z)$. Applying rule 7, one obtains $x \leftrightarrow y \leftrightarrow z$, an equivalency clause to be added into $\mathcal{F}$. Rule 6 is also used in this way to deduce new equivalent literals.

It can be shown from its own construction that DIMACS pret* problem, although hard for a classical implementation of DP procedure, is solved in linear time by repeatedly adding new ternary equivalency clauses into $\mathcal{F}$.

$EqSatz$: An Equivalency Reasoning Enhanced DP

We implement the six inference rules defined in section 2 into a highly optimized DP procedure called $Satz$ (Li and Anbulagan 1997). $Satz$ consists of a fast unit propagator and a powerful look-ahead heuristic based on unit propagation to select the next branching variable to maximize the reduction of search space when branching. Equivalency reasoning enhanced $Satz$ is called $EqSatz$ and is sketched in Figure 1. Figure 2 shows the implementation of equivalency reasoning. Figure 3 shows the branching rule of $EqSatz$.

[1] ftp://dimacs.rutgers.edu/pub/challenge/sat

The equivalency clauses contained in the input formula are in CNF form. Rules 2 and 3 are performed by unit propagation, so equivalency reasoning begins by detecting equivalent literals produced by unit propagation. If l_1 is equivalent to l_2, it is substituted by l_2. Note that the substitution may produce a unit clause in case $l_1 \vee l_2$ is a clause in $\mathcal{F}$. Property 1 is used to rewrite equivalency clauses in an appropriate form to apply the inference rules.

Equivalency reasoning is naturally integrated in the heuristic of $Satz$. Given a free variable x, $Satz$ examines x by respectively adding two unit clauses x and $\bar{x}$ into $\mathcal{F}$ and makes two experimental unit propagations to see the impact of branching on x. Following this line, $EqSatz$ performs an experimental equivalency reasoning after each experimental unit propagation to look further forward and to add new equivalency clauses into $\mathcal{F}$ using rules 6 and 7.

Like $Satz$, $EqSatz$ tries to branch to the variable allowing to maximize the reduction of search space by taking equivalency reasoning into account and uses three functions to estimate the reduction of search space.

Procedure $EqSatz(\mathcal{F})$
Begin
  if $\mathcal{F}$ is empty, return "satisfiable";
  $\mathcal{F}$:=UnitPropagation($\mathcal{F}$);
  $\mathcal{F}$:=Equivalency_Reasoning($\mathcal{F}$);
  if $\mathcal{F}$ contains an empty clause,
  return "unsatisfiable".

  /* branching rule */
  select a variable x in $\mathcal{F}$.
  If $EqSatz(\mathcal{F} \cup \{x\})$ is "satisfiable"
  then return "satisfiable", otherwise
  return the result of $EqSatz(\mathcal{F} \cup \{\bar{x}\})$.
End.

procedure UnitPropagation($\mathcal{F}$)
Begin
  While there is no empty clause and a unit
  clause l exists, satisfy l and simplify $\mathcal{F}$.
  Return $\mathcal{F}$.
End.

Figure 1: The DP Procedure $EqSatz$

The first function is $nb\_fixed\_vars(\mathcal{F}_1, \mathcal{F}_2)$ giving the number of variables of $\mathcal{F}_2$ instantiated in $\mathcal{F}_1$, and the second is $nb\_eq\_pairs(\mathcal{F}_1, \mathcal{F}_2)$ giving the number of equivalent literals of $\mathcal{F}_2$ substituted in $\mathcal{F}_1$. The motivation is that the instantiation of a variable or a new deduced literal equivalence such as $l_3 \leftrightarrow l_4$ cuts the search space in half. The third function $nb\_binary\_clauses(\mathcal{F}_1, \mathcal{F}_2)$ ($nb\_bin\_cls(\mathcal{F}_1, \mathcal{F}_2)$ in short) is defined following the same line. But the consideration is somewhat more complicated.

If $\mathcal{F}$ has n variables, it has 2^n possible solutions. A binary clause removes 2^{n-2} solutions. But generally two binary clauses together do not remove 2^{n-1} solutions. In fact two binary clauses sharing an identical literal or having no

common variable such as $x_1 \vee x_2$ and $x_1 \vee x_3$ or $x_1 \vee x_2$ and $\bar{x}_3 \vee x_4$ remove $2^{n-1} - 2^{n-3}$ and $2^{n-1} - 2^{n-4}$ solutions, respectively. Only two binary clauses sharing a complementary literal such as $x_1 \vee x_2$ and $\bar{x}_1 \vee x_3$ remove 2^{n-1} solutions. Clearly binary clauses sharing complementary literals remove many more solutions and have more chances to lead to a dead-end where all solutions are removed. So a DP procedure should branch next on the variable generating subproblems having more binary clauses sharing complementary literals.

$$nb\_bin\_cls(\mathcal{F}_1, \mathcal{F}_2) = \sum_{l \vee l' \ is \ in \ \mathcal{F}_1 \ but \ not \ in \ \mathcal{F}_2} [f(\bar{l}) + f(\bar{l'})]$$

where $f(\bar{l})$ is the number of binary occurrences of $\bar{l}$ in $\mathcal{F}_2$.

Procedure Equivalency_Reasoning($\mathcal{F}$)
Begin
  $\mathcal{F}$:=Set_Equivalence_by_CNF_Clauses($\mathcal{F}$);
  repeat
    /* Rule 4 */
    while there is an equivalency clause containing two identical literals such as $l_1 \leftrightarrow l_1 \leftrightarrow l_3$ but no empty clause do
    begin
      satisfy l_3 and simplify $\mathcal{F}$;
      $\mathcal{F}$:=Set_Equivalence_by_CNF_Clauses($\mathcal{F}$);
    end;
    /* Rule 5 */
    while there are two equivalency clauses containing two identical literals such as $l_1 \leftrightarrow l_2 \leftrightarrow l_3$ and $l_1 \leftrightarrow l_2 \leftrightarrow l_4$ but no empty clause do
      if $l_3 = \neg l_4$ then add an empty clause into $\mathcal{F}$
      else
      begin
        $\mathcal{F}$:=Substitute(l_3, l_4, $\mathcal{F}$);
        $\mathcal{F}$:=Set_Equivalence_by_CNF_Clauses($\mathcal{F}$);
      end
  until an empty clause is produced
      or no change happens in $\mathcal{F}$
End.

procedure Set_Equivalence_by_CNF_Clauses($\mathcal{F}$)
Begin
  while $\mathcal{F}$ contains two binary CNF clauses such as $l_1 \vee \bar{l}_2$ and $\bar{l}_1 \vee l_2$ but no empty clause do
    $\mathcal{F}$:=Substitute(l_1, l_2, $\mathcal{F}$);
  return $\mathcal{F}$
End.

procedure Substitute($l_1, l_2, \mathcal{F}$)
Begin
  substitute all occurrences of $l_1(\bar{l}_1)$ by $l_2(\bar{l}_2)$;
  $\mathcal{F}$:=UnitPropagation($\mathcal{F}$); return $\mathcal{F}$;
End.

Figure 2: The Procedure Equivalency Reasoning

Solving the Challenge DIMACS 32-bit Parity Problem

$EqSatz$ was originally motivated by the challenge DIMACS 32-bit parity problem formulated by Selman et al. (1997) at IJCAI'97. To the best of our knowledge, $EqSatz$ is the *only* procedure which is able to solve all the ten par32* instances in reasonable time.

Table 1 shows the performance of $EqSatz$ on the challenge problem. As in the next section, #cls and #eq_cls[2] respectively denote the total number of clauses in the input CNF formula and the number of ternary equivalency clauses, a ternary equivalency clause being counted as 4 clauses in #cls. As can be seen, All instances contain a large EQ part.

Equivalency reasoning makes $EqSatz$ substantially faster than $Satz$ on the challenge parity problem. In the next section we show that it is sufficiently powerful to make a DP procedure able to solve other problems.

For each free variable x do
let $\mathcal{F}'$ and $\mathcal{F}''$ be two copies of $\mathcal{F}$
Begin
  $\mathcal{F}'$:= UnitPropagation($\mathcal{F}' \cup \{x\}$);
  $\mathcal{F}''$:= UnitPropagation($\mathcal{F}'' \cup \{\bar{x}\}$);
  $\mathcal{F}'$:=Equivalency_Reasoning($\mathcal{F}'$);
  $\mathcal{F}''$:=Equivalency_Reasoning($\mathcal{F}''$);
  If both $\mathcal{F}'$ and $\mathcal{F}''$ contain an empty clause
    then return "$\mathcal{F}$ is unsatisfiable".
  else if $\mathcal{F}'$ contains an empty clause then
        $x := 0$, $\mathcal{F} := \mathcal{F}''$
  else if $\mathcal{F}''$ contains an empty clause then
        $x := 1$, $\mathcal{F} := \mathcal{F}'$;
  else
  begin
    /* Rule 6 */
    for all $l_3 \leftrightarrow l_4 \in \mathcal{F}' \wedge \mathcal{F}''$
        $\mathcal{F}$:=Substitute(l_3,l_4,$\mathcal{F}$);
    /* Rule 7 */
    for all $l_3 \leftrightarrow l_4 \in \mathcal{F}'$ and $\bar{l}_3 \leftrightarrow l_4 \in \mathcal{F}''$
        add $x \leftrightarrow l_3 \leftrightarrow l_4$ into $\mathcal{F}$.
    $\mathcal{F}$:=Equivalency_Reasoning($\mathcal{F}$);
    if $\mathcal{F}$ contains an empty clause then
        return "$\mathcal{F}$ is unsatisfiable".
    let $w(l)$ denote the weight of literal l,
    $w(x) := nb\_fixed\_vars(\mathcal{F}', \mathcal{F}) + nb\_eq\_pairs(\mathcal{F}', \mathcal{F})$
        $+nb\_bin\_cls(\mathcal{F}', \mathcal{F})/2$
    $w(\bar{x}) := nb\_fixed\_vars(\mathcal{F}'', \mathcal{F}) + nb\_eq\_pairs(\mathcal{F}'', \mathcal{F})$
        $+nb\_bin\_cls(\mathcal{F}'', \mathcal{F})/2$;
  end
End;
For each free variable x do
  $H(x) := w(\bar{x}) * w(x) * 1024 + w(\bar{x}) + w(x)$;
Branching on the free variable x such that $H(x)$ is the greatest.

Figure 3: The Branching Rule of $EqSatz$

[2] $EqSatz$ inherits from Satz a preprocessing of the input instance searching and adding resolvents of length ≤ 3, which may remove some equivalency clauses. #eq_cls denotes the number of remaining ternary equivalency clauses.

Table 1: Run time and search tree size (t_size) of $EqSatz$ on the challenge DIMACS 32-bit parity problem

Instance	#var	#cls	#eq_cls	time	t_size
par32-1-c	1315	5254	1097	1133	3672
par32-2-c	1303	5206	1085	50	209
par32-3-c	1325	5294	1107	3972	15123
par32-4-c	1333	5326	1115	793	1488
par32-5-c	1339	5350	1121	9265	38348
par32-1	3176	10277	1097	989	3089
par32-2	3176	10253	1085	241	651
par32-3	3176	10297	1107	8899	23827
par32-4	3176	10313	1115	827	2885
par32-5	3176	10325	1121	11855	35133

Other Experimental Results

We use three separate benchmarks involving equivalency clauses in the literature to evaluate the impact of equivalency reasoning in a DP procedure and to compare the performance of $EqSatz$ with 4 state-of-the-art DP procedures on the same instances: DIMACS pret* problem, Massacci's Data Encryption Standard (DES) problem[3], and Biere et al.'s Bounded Model Checking (BMC) problems[4].

The four state-of-the-art DPs compared are Sato (Zhang 1997), Grasp (Silva and Sakallah 1996), Relsat (Bayardo and Schrag 1997) and Satz. Sato, Grasp and Relsat use both look-ahead techniques such as unit propagation and variable ordering heuristics for branching and look-back techniques such as intelligent backtracking and learning, while Satz uniquely uses look-ahead techniques. So the comparison between $EqSatz$ and Satz in the experimentation illustrates the impact of equivalency reasoning and the comparison of $EqSatz$ with Sato, Grasp[5], and Relsat might be considered as a comparison between look-back techniques and equivalency reasoning on the instances involving equivalency clauses.

Performance on DIMACS pret* problem

The problem was contributed by Pretolani, inspired by Urquhart's construction (Urquhart 1987). We modify some constants in the generator available at the same site to generate larger instances. These unsatisfiable instances are uniquely composed of equivalency clauses.

$EqSatz$ solves Pretolani's problem in linear time. Table 2 shows the performance of the 5 DP procedures. Note that though Sato, Grasp and Relsat are substantially faster than Satz on these instances, it seems that they still have an exponential behavior.

[3]available from http://www.uni-koblenz.de/~massacci

[4]available from http://www.cs.cmu.edu/~modelcheck

[5]when solving some instances, Grasp is stopped before 2 hours of run time because of memory shortage or resource exceeded. Its runtime is marked by "?"

Table 2: run time of Sato, Grasp, Relsat, Satz and $EqSatz$ on DIMACS pret* problem

#vars	300	450	600	750	1500	3000
#eq_cls	200	300	400	500	1000	2000
Grasp	9	31	82	166	1742	> 7200
Sato	28	37	102	> 7200	-	-
Relsat	1	2	4	10	72	696
Satz	> 7200	-	-	-	-	-
$EqSatz$	0	0	0	0	0	0

Performance on DES instances

DES instances are contributed by Massacci (1999). These are SAT-encoding of cryptographic key search problem and contain equivalency clauses from 3 rounds. We only report on 3 round instances, since to our knowledge no SAT solver solves instances of more than 3 rounds. The original instances involve a huge number of variables with no clauses. So we compact them by making unit resolution and pure literal elimination and renaming the variables to be contiguous. The instances after the simplification are listed in table 3. All instances are satisfiable.

Table 3: DES instances

Instance	#var	#clause	#eq_cls
cnf-r3-b1-k1.1	1461	8966	48
cnf-r3-b1-k1.2	1450	8891	48
cnf-r3-b2-k1.1	2855	17857	96
cnf-r3-b2-k1.2	2880	17960	96
cnf-r3-b3-k1.1	4255	26778	144
cnf-r3-b3-k1.2	4418	27503	144
cnf-r3-b4-k1.1	5679	35817	192
cnf-r3-b4-k1.2	5721	35963	192

Table 4 displays the performance of the 5 DPs on Massacci's DES 3-round instances. $EqSatz$ is one of the fastest procedures to solve these instances and is substantially faster than Satz, illustrating the impact of equivalency reasoning to solve these instances even when there are very few equivalency clauses.

Table 4: Run time on DES instances. The name of each instance should be preceded by "cnf-r3"

Instance	Sato	Grasp	Relsat	Satz	$EqSatz$
b1-k1.1	871	?	1080	> 7200	995
b1-k1.2	> 7200	?	454	> 7200	1023
b2-k1.1	3	170	18	946	1276
b2-k1.2	> 7200	183	22	1468	629
b3-k1.1	> 7200	96	37	32	11
b3-k1.2	> 7200	113	44	101	11
b4-k1.1	> 7200	77	45	357	17
b4-k1.2	> 7200	67	48	> 7200	18

Performance on BMC instances

BMC problems are contributed by Biere et al. (1999) and arise from (bounded) model checking. All instances are unsatisfiable. We select the most difficult barrel* and queuein-

var* instances and representative half longmult* instances. The selected instances are listed in Table 5.

Table 5: BMC instances

Instance	#var	#clause	#eq_cls
barrel5	1407	5383	870
barrel6	2306	8931	1476
barrel7	3523	13765	2310
barrel8	5106	20083	3408
barrel9	8903	36606	6408
longmult1	791	2335	29
longmult3	1555	4767	87
longmult5	2397	7431	145
longmult7	3319	10335	203
longmult9	4321	13479	261
longmult11	5403	16863	319
longmult13	6565	20487	377
longmult15	7807	24351	435
queueinvar10	886	5622	51
queueinvar12	1112	7335	53
queueinvar14	1370	9313	55
queueinvar16	1168	6496	75
queueinvar18	2081	17368	70
queueinvar20	2435	29671	72

Table 6 displays the performance of the 5 DPs on BMC instances. $EqSatz$ is substantially faster than the 4 other DPs on the barrel* and queueinvar* instances. On barrel* instances containing a more important EQ part, $EqSatz$ finds the inconsistency by equivalency reasoning without branching. On longmult* instances, only Sato is slightly faster than $EqSatz$.

Table 6: Run time on BMC instances. Each name mult* should be preceded by "long" and each name invar* by "queue"

Instance	Sato	Grasp	Relsat	Satz	$EqSatz$
barrel5	25	?	264	293	0
barrel6	281	?	4428	2461	1
barrel7	530	?	> 7200	57	2
barrel8	726	?	> 7200	5	3
barrel9	> 7200	?	> 7200	> 7200	7
mult1	0	0	0	0	0
mult3	41	1	1	0	1
mult5	181	64	27	11	13
mult7	348	?	3402	331	274
mult9	733	?	> 7200	1948	1681
mult11	1110	?	> 7200	4371	3050
mult13	1916	?	> 7200	6965	3662
mult15	2646	?	> 7200	> 7200	4876
invar10	15	27	20	12	3
invar12	97	52	31	54	5
invar14	576	101	162	250	10
invar16	1398	109	93	1017	9
invar18	> 7200	?	257	8	9
invar20	> 7200	?	834	13	14

Discussion and Related Work

Equivalency clauses constitute a major obstacle to DP procedure. For example, while the complexity of Satz on the most difficult random 3-SAT instances appears to be $O(2^{n/21})$, its complexity on DIMACS pret* problem is $O(2^{n/3})$. However, equivalency clauses realize a common structure in the SAT-encoding of many hard real-world problems. Massacci (1999) noticed in his SAT-encoding of cryptographic key search problem that each round is separated from the next round by a level of equivalency clauses and most constraints are in form of equivalency clauses. He also noted that the problem becomes hard for current AI techniques as soon as equivalency clauses start to appear.

Learning and intelligent backtracking are general CSP look-back techniques effective for many structured problems. However they appear to be less effective on the special structure of equivalency clauses than the specialized look-ahead technique equivalency reasoning. For example, although they make Sato, Grasp and Relsat substantially faster than Satz on DIMACS pret* problem, equivalency reasoning makes the complexity of $EqSatz$ linear on this problem.

For instances containing both EQ part and CNF part, equivalency reasoning plays its role even when the EQ part is very small. For example, $EqSatz$ is significantly faster than Satz and competes with the most efficient look-back enhanced DP on DES instances where the EQ part is only roughly 2% of the whole formula. On DIMACS 32-bit parity problem and BMC barrel* problem containing a large EQ part, $EqSatz$ is substantially faster than other DPs.

Neither equivalency reasoning nor CSP look-back techniques allow a DP to solve Massacci's 4-round DES instances in reasonable time. The reason appears to be that these instances have other structures. We believe that combining equivalency reasoning and look-back techniques is promising to solve these instances.

Warners and Van Maaren (1998) proposed an approach to solve the instances only having EQ part or whose EQ part is much larger than their CNF part. Their approach gave the first method able to solve the 5 DIMACS 32-bit parity instances par32-i-c in which more than 80% of clauses are equivalency clauses. The originality is the elimination of equivalency clauses.

Given a set of equivalency clauses, Warners and Van Maaren select an equivalency clause of length k, $x_1 \leftrightarrow l_2 \leftrightarrow \ldots \leftrightarrow l_k$, write it as $x_1 \equiv l_2 \leftrightarrow \ldots \leftrightarrow l_k$ and substitute in all other equivalency clauses the occurrence of $(\bar{x}_1)\ x_1$ by $(\neg) l_2 \leftrightarrow \ldots \leftrightarrow l_k$, increasing in general the length of these clauses (by $k-2$ in the worst case). x_1 is called *dependent variable*.

For a formula $\mathcal{F}$ having no CNF part such as DIMACS pret* and dubois*, the selected equivalency clause can be easily satisfied since its dependent variable doesn't occur elsewhere after the above substitution, so that it is removed from $\mathcal{F}$. By repeatedly removing equivalency clauses $\mathcal{F}$ is solved in polynomial time. For problems containing both EQ and CNF part, the substitution is limited in the EQ part

and the selected equivalency clause cannot be removed if its dependent variable occurs in the CNF part. Otherwise the selected equivalency clause is removed.

Warners and Van Maaren's algorithm is divided into two phases to solve an instance containing both CNF part and EQ part. In the first phase the above substitution operation is applied to the EQ part to eliminate as many equivalency clauses as possible. For example, the five par32-i-c instances contain more than 1085 equivalency clauses, but only 218 equivalency clauses remain after the first phase (i.e. there are 218 dependent variables occurring in the CNF part). In the second phase an adapted DP procedure is executed on the CNF part together with the 218 equivalency clauses. So the first phase might be considered as a special processing in the root of a search tree, which is to be compared with the equivalency reasoning of $EqSatz$ in *every* node of a search tree.

Warners and Van Maaren's algorithm is actually substantially faster than $EqSatz$ on the five par32-i-c instances which are solved in at most 50 seconds (Warners 1999). The efficiency seems due to the fact that few (i.e. 218) equivalency clauses remain after the first phase for their DP procedure. However, when the CNF part of a formula is larger and contains more dependent variables, less equivalency clauses can be removed. In this case $EqSatz$ outperforms Warners and Van Maaren's algorithm. In fact, the five par32-i instances, the DES instances and the BMC instances are in this case. For example, for the par32-1 instance which contains 1548 equivalency clauses and 4085 other CNF clauses, 1048 (longer) equivalency clauses remain after the first phase (Warners 1999). To the best of our knowledge, $EqSatz$ is the *only* procedure which is able to solve all the five par32-i instances in reasonable time.

Conclusion

Equivalency clauses represent a common structure in the SAT-encoding of many hard real-world problems. They constitute a major obstacle to DP procedure. We have integrated an equivalency reasoning into Satz to solve problems containing equivalency clauses. Equivalency reasoning allows to remedy the ineffectiveness of unit propagation on equivalency clauses. Integrated in the branching rule, it makes the branching rule more precise and more powerful to maximize the reduction of search space when branching. Finally the new equivalency clauses added into $\mathcal{F}$ by equivalency reasoning also reduce considerably the search space.

Our approach makes DP able to solve one of the ten challenge problems of propositional reasoning formulated by Selman et al. and many other real-world problems. The experimental results suggest that equivalency reasoning is more effective than the general CSP look-back techniques on the special structure of equivalency clauses.

Acknowledgments

We thank Daniel Le Berre for informing us BMC problems, Patrice Seebold and anonymous referees for their comments which helped improve this paper.

References

Bayardo Jr. R.J., Schrag R.C., Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In proceedings of AAAI-97, Providence, Rhode Island, July 1997.

Biere A., Cimatti A., Clarke E., Zhu Y., Symbolic Model Checking without BDDs. In proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS. Springer-Verlag, 1999.

Cook S.A., The Complexity of Theorem Proving Procedures. In *3rd ACM Symp. on Theory of Computing*, pages 151-158, Ohio, 1971.

Crawford J.M., Kearns M.J., Schapire R.E., The Minimal Disagreement parity problem as a hard satisfiability problem. Draft version 1995.

Davis M., Logemann G., Loveland D., A Machine Program for Theorem Proving. In Common. ACM 5, 1962, pp. 394-397.

Li C.M., Anbulagan, Heuristics Based on Unit Propagation for Satisfiability Problems. In Proceedings of IJCAI-97, ISBN 1-55860-480-4, Page 366-371, Nagoya, Japan, August 1997.

Massacci F., Using Walk-SAT and Rel-SAT for Cryptographic Key Search. In Proceedings of IJCAI-99, Page 290-295.

Selman B., Kautz H., McAllester D., Ten challenges in propositional reasoning and search. In Proc. of IJCAI-97, ISBN 1-55860-480-4, Nagoya, Japan, August 1997.

Silva J. P. M., Sakallah K. A., Conflict Analysis in Search Algorithms for Propositional Satisfiability. In Proc. of the Int. Conf. on Tools with Artificial Intelligence, November 1996.

Urquhart A., Hard examples for resolution. Journal of the ACM, 34(1):209-219, January 1987.

Warners J.P., Van Maaren H., A two phase algorithm for solving a class of hard satisfiability problems. Operations research letters 23 (1998) 81-88.

Warners J.P., Personal communication, June 1999.

Zhang H., An efficient propositional prover. In Proc. of int' Conf. on Automated Deduction, pp272-275, July 1997.