Abstract Model Generation for Preprocessing Clause Sets

Miyuki Koshimura, Mayumi Umeda, and Ryuzo Hasegawa

Kyushu University, 6-1 Kasuga-koen, Kasuga, Fukuoka, 816-8580 Japan
{koshi,umeda,hasegawa}@ar.is.kyushu-u.ac.jp

Abstract. Abstract model generation refers to model generation for abstract clause sets in which arguments of atoms are ignored. We give two abstract clause sets which are obtained from normal clause sets. One is for checking satisfiability of the original normal clause set. Another is used for eliminating unnecessary clauses from the original one. These abstract clause sets are propositional, i.e. decidable. Thus, we can use them for preprocessing the original one.

1 Introduction

The use of abstraction seems to be helpful in many subfields of artificial intelligence [10,7,4,3,2]. The most common use of abstraction in theorem proving has been to abstract the problem, to prove its abstracted version, and then to use the structure of the resulting proof as guides in searching for the original problem. This assumes that the structure of the abstract proof is similar to that of the original problem. The most common approach is to integrate the abstract proving into the deduction process by specifying clause selection functions that imitate the abstract proof. On the other hand, there is another approach which uses the abstract proving as a preprocessing step in the (ground) prover [8].

The benefit of preprocessing a set S of clauses can be large. In the extreme S may be solved in the preprocessing stage. In this paper, we use model generation [6,5] as a procedure for preprocessing S rather than proving S. We apply model generation to abstractions of S. We present two types of abstraction; *c-abstraction* and *d-abstraction*. In these abstractions, we abstract away all arguments from atoms. Thus, abstract clause sets are propositional.

S is satisfiable if its d-abstraction is satisfiable. In this case, we determine its satisfiability without proving S itself. If a clause in S contains an atom whose abstraction is not in the model of c-abstraction of S, the clause is unnecessary for checking unsatisfiability. Thus, the clause can be eliminated.

This c-abstraction based elimination is a kind of simplification which simplifies a set of clauses. Its effect is parallel to that of a simplification operation eliminating pure literals [15]. However, their strength is not comparable. That is, the former can eliminate more clauses than the latter does in some cases, and vice versa. We evaluate effects of abstract model generation for preprocessing with all CNF problems in the TPTP problem library.

2 Model Generation

Throughout this paper, a *clause* $\neg A_1 \vee \ldots \vee \neg A_m \vee B_1 \vee \ldots \vee B_n$ is represented in implicational form: $A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$ where A_i $(1 \leq i \leq m)$ and B_j $(1 \leq j \leq n)$ are atoms; the left hand side of "$\rightarrow$" is said to be the *antecedent*; and the right hand side of "$\rightarrow$" the *consequent*.

A clause is said to be *positive* if its antecedent is *true* $(m = 0)$, and *negative* if its consequent is $false$ $(n = 0)$; otherwise it is *mixed* $(m \neq 0, n \neq 0)$. A clause is said to be *violated* under a set M of ground atoms if with some ground substitution σ the following condition holds: $\forall i(1 \leq i \leq m)A_i\sigma \in M \wedge \forall j(1 \leq j \leq n)B_j\sigma \notin M$.

A model generation proof procedure is sketched in Fig. 1. The procedure MG takes a partial interpretation Mc (model candidate) and a set of clauses S to be proven, and builds a (sub)proof-tree of S.

A leaf labeled with $\top$ tells us that a model of S has been found as a current model candidate. If every leaf of the constructed proof-tree is labeled with $\perp$, S is unsatisfiable; otherwise S is satisfiable. In the latter case, at least one leaf is labeled with $\top$ or at least one branch grows infinitely.

```
procedure $MGTP(S) : P$; /* Input($S$):Clause set, Output($P$):Proof-tree of $S$ */
    return($MG(\emptyset, S)$);

procedure $MG(Mc, S) : P$;/* Input($Mc$): Model candidate */
 1. (Model rejection) If a negative clause $(A_1 \wedge \ldots \wedge A_m \rightarrow false) \in S$ is
    violated under $Mc$ with a ground substitution $\sigma$, return$\langle \perp \rangle$
 2. (Model extension) If a positive or mixed clause $(A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n) \in S$
    is violated under $Mc$ with a ground substitution $\sigma$,
    return $\langle$ [tree] $\rangle$
    where $P_i = MG(Mc \cup \{B_i\sigma\}, S)$ $(1 \leq i \leq n)$.
 3. (Model finding) If neither 1 nor 2 is applicable, return $\langle \top \rangle$;
```

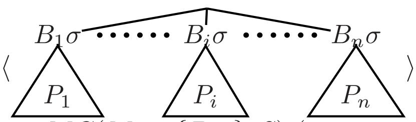

Fig. 1. Model generation procedure

3 Abstract Clauses

An *abstract atom* of $P(t_1, \ldots, t_n)$ is an atom P. That is, the abstract atom abstracts away its arguments. Henceforth, we will use capital letters $A, B, A_1, B_1, \ldots$ as denoting normal atoms, and small letters $a, b, a_1, b_1, \ldots$ as denoting abstract atoms corresponding to the capital letters.

A *d-abstract clause* of $A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$ is a clause $a_1 \wedge \ldots \wedge a_m \rightarrow b_1 \vee \ldots \vee b_n$. A set of d-abstract clauses obtained from a normal clause set S by replacing normal clauses with d-abstract ones is denoted by $d\_abs(S)$.

Theorem 1. *Let S be a set of clauses. If $d\_abs(S)$ is satisfiable, then S is satisfiable.*

$d\_abs(S)$ is a set of propositional clauses, so, checking its satisfiability is decidable, while checking satisfiability of S is generally undecidable[1].

Example 1 (D-abstraction). Let $S = \{p(x) \wedge q(x) \wedge s(y) \rightarrow false,\ p(x) \wedge r(x) \rightarrow s(f(x)),\ q(x) \rightarrow r(x) \vee s(f(x)),\ p(x) \rightarrow q(x) \vee r(y),\ true \rightarrow p(a)\}$, then $d\_abs(S) = \{p \wedge q \wedge s \rightarrow false,\ p \wedge r \rightarrow s,\ q \rightarrow r \vee s,\ p \rightarrow q \vee r,\ true \rightarrow p\}$.

$d\_abs(S)$ has a model $\{p, r, s\}$ and thus is satisfiable. Therefore, we conclude S is satisfiable.

C-abstract clauses of $A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$ are n clauses $a_1 \wedge \ldots \wedge a_m \rightarrow b_1,\ a_1 \wedge \ldots \wedge a_m \rightarrow b_2,\ \cdots,$ and $a_1 \wedge \ldots \wedge a_m \rightarrow b_n$. Note that there is no c-abstract clause for a negative clause $A_1 \wedge \ldots \wedge A_m \rightarrow false$.

A set of c-abstract clauses obtained from a normal clause set S by replacing normal clauses with c-abstract ones is denoted by $c\_abs(S)$. Note that negative clauses are eliminated in $c\_abs(S)$ and all clauses in $c\_abs(S)$ are Horn clauses. Therefore, we obtain a unique model of $c\_abs(S)$ with the model generation procedure.

A clause $A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n$ is *relevant* to a set $\mathcal{A}$ of abstract atoms if $\forall i(1 \leq i \leq m)(a_i \in \mathcal{A})$, otherwise, *irrelevant*. If a clause $C(\in S)$ is used for model extension or rejection in the model generation procedure on S, C is relevant to the model of $c\_abs(S)$. Thus, we obtain the following lemma.

Lemma 1. *Let S be a set of clauses, P be a proof tree of S, M be a model of $c\_abs(S)$, and $C = A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n \in S$ be a clause used for model extension or rejection in P. Then, $\forall i(1 \leq i \leq m) a_i \in M$ where a_i is the abstract atom of $A_i (1 \leq i \leq m)$. That is, C is relevant to M.*

Proof. Let $C^k = A_1^k \wedge \ldots \wedge A_m^k \rightarrow B_1^k \vee \ldots \vee B_n^k \in S$ be a clause used for the k-th model extension or rejection in P. We can easily show the following property by induction on k: $\forall i(1 \leq i \leq m) a_i^k \in M \wedge \forall j(1 \leq j \leq n) b_j^k \in M$ where a_i^k is the abstract atom of $A_i^k (1 \leq i \leq m)$ and b_j^k is the abstract atom of $B_j^k (1 \leq j \leq n)$. This property implies the lemma. □

The lemma says that if a clause $C(\in S)$ is irrelevant to the model of $c\_abs(S)$, then C is never used for model extensions or rejections in the model generation procedure on S. Therefore, we ignore irrelevant clauses when we apply the model generation procedure on S.

[1] $d\_abs(S)$ is exactly the same as the propositional abstraction proposed by Plaisted [7] and thus folklore. But, it is still interesting to see experimental data on all satisfiable problems from the TPTP library.

Theorem 2. *If the model generation determines that a set S of clauses is unsatisfiable, then it also determines that $S \backslash IR(S)$ is unsatisfiable, where $IR(S) = \{C|C$ is irrelevant to $M\}$ and M is a model of $c\_abs(S)$.*

The model generation is a sound and complete proof procedure [1], therefore, we obtain the corollary.

Corollary 1. *Let S be a set of clauses. Then, S is unsatisfiable iff $S \setminus IR(S)$ is unsatisfiable.*

By proving $S \setminus IR(S)$ instead of S, we can diminish its execution cost if $IR(S) \neq \emptyset$. If $S \setminus IR(S) = \emptyset$, we conclude that S is satisfiable. When $IR(S) = \emptyset$, we may decrease the number of clauses in S by applying the same consideration on the set $CON(S) = \{B_1 \wedge \ldots \wedge B_n \rightarrow A_1 \vee \ldots \vee A_m \mid (A_1 \wedge \ldots \wedge A_m \rightarrow B_1 \vee \ldots \vee B_n) \in S\}$ which is obtained from the contrapositive set of S by reversing every literal polarity. Thus, we obtain a process which eliminates unnecessary clauses in S:

(1) Let S be a set of clauses.
(2) $i = 0,\ S_0 = S$.
(3) $S_{i+1} = S_i \setminus IR(S_i)$, $i = i + 1$.
(4) If $i = 1$ or $S_i \neq S_{i-1}$, then $S_i = CON(S_i)$ and goto (3).
(5) If i is an even number, then $S_i = CON(S_i)$.

We stop the process when it reaches a fixpoint gotten as $S_i = S_{i-1}$. Then, we try to prove the final S_i instead of S.

Example 2 (C-abstraction). Let $S(= S_0)$ be a set of 6 clauses from $C1$ to $C6$:

$$\begin{array}{lll} C1: r(x) \rightarrow false & C2: v(x) \rightarrow r(x) & C3: s(x) \rightarrow r(x) \\ C4: q(x) \rightarrow s(x) \vee u(x) & C5: p(x) \rightarrow q(x) & C6: true \rightarrow p(a) \end{array}$$

Then, c-abstraction $c\_abs(S_0)$ is a set of the following clauses:

$$\begin{array}{lll} C2_1: v \rightarrow r & C3_1: s \rightarrow r & C4_1: q \rightarrow s \\ C4_2: q \rightarrow u & C5_1: p \rightarrow q & C6_1: true \rightarrow p \end{array}$$

We obtain the model $\{p,\ q,\ u,\ s,\ r\}$ of $c\_abs(S_0)$ with model generation. The clause $C2$ is irrelevant to this model and thus eliminated. So, $S_1 = \{C1, C3, C4, C5, C6\}$, then $S_1 = CON(S_1) = \{C1^C, C3^C, C4^C, C5^C, C6^C\}$ where

$$\begin{array}{lll} C1^C: true \rightarrow r(x) & C3^C: r(x) \rightarrow s(x) & C4^C: s(x) \wedge u(x) \rightarrow q(x) \\ C5^C: q(x) \rightarrow p(x) & C6^C: p(a) \rightarrow false & \end{array}$$

Next, we obtain the model $\{r,\ s\}$ of $c\_abs(S_1)$. Therefore, $C4^C$, $C5^C$, and $C6^C$ are irrelevant to this model and thus eliminated. So, $S_2 = \{C1^C, C3^C\}$, then $S_2 = CON(S_2) = \{C1, C3\}$. We continue this process until no clause is eliminated. Finally, S_3 becomes an empty set. Thus, we conclude S is satisfiable.

4 Experimental Results

The method is implemented on top of a constraint logic programming system B-Prolog [17]. We use all 5522 CNF problems in the TPTP problem library version 2.7.0 [13,14]. We remove equality axioms using the tptp2X utility (distributed with TPTP) with option "`-t rm_equality:rsftp`" from each problem if any. The problems were run on a DELL computer (Mobile Intel Pentium III 650MHz CPU, 512MB memory, Linux 2.6.0).

If a problem S contains equality which is represented using the `equal/2` predicate in TPTP, we simply add a positive unit clause "$true \rightarrow equal$" to $d\_abs(S)$ and $c\_abs(S)$ before preprocessing. In other words, we assume that all individuals are equal in d-abstraction and the `equal/2` predicate is relevant in c-abstraction.

4.1 D-Abstraction: Checking Satisfiability

In 766 satisfiable first-order problems, 223 problems are determined as satisfiable with their d-abstract clause sets, within one second for each. Table 1 (a) shows the number of problems solved by d-abstraction for each problem domain in TPTP. The first column shows domain names, and the second column shows the number of problems solved and the number of satisfiable first-order problems in that domain. For example, there are 17 satisfiable first-order problems in the BOO domain. Among them, 4 problems are solved by d-abstraction. 45 % of 223 problems are in the SYN category and 19 % are in the NLP category. Table 1 (b) shows similar information for every problem rating[2]. The effectiveness of d-abstraction seems to be independent of the problem domains and ratings.

4.2 C-Abstraction: Eliminating Unnecessary Clauses

In 5522 CNF problems, 725 problems are reduced by c-abstraction based elimination. For the ALG, COM, FLD, GRA, HEN, HWC, KRS, LDA, RNG, ROB, SWC, and TOP categories, no problem is reduced. The average preprocessing time is 3.75 seconds for 725 problems. More than 90% problems are reduced within one second for each, while 35 problems need more than ten seconds for reducing. All these 35 problems are in the SYN category and consist of more than 1000 clauses.

Table 2 (a) shows the numbers of problems reduced with c-abstraction for each problem domain. For example, there are 83 CNF problems in the HWV category. 31 problems of them are reduced. For the NLP and SYN categories, more than half of the problems are reduced.

[2] In the TPTP distribution, each problem file consists of a header part and a body part. The header part contains information about problem. The rating filed is in the header part. The rating gives the difficulty of the problem. It is a real number in the range 0.0 to 1.0, where 0.0 means that the problem is easy and 1.0 means that the problem is hard.

Table 1. Numbers of problems solved by d-abstraction

(a) Domain

ALG	0/2	GRA	0/0	LCL	10/44	RNG	5/10
ANA	2/2	GRP	12/75	LDA	0/0	ROB	2/5
BOO	4/17	HAL	0/0	MGT	2/11	SET	4/12
CAT	6/10	HEN	3/3	MSC	1/2	SWC	0/1
COL	0/6	HWC	2/2	NLP	42/236	SWV	3/9
COM	0/0	HWV	6/8	NUM	4/7	SYN	100/216
FLD	0/0	KRS	2/8	PLA	2/2	TOP	1/19
GEO	0/17	LAT	8/22	PUZ	2/20		

(b) Rating

0.00	91/232	0.67	0/80
0.14	41/79	0.71	18/29
0.17	4/33	0.83	0/1
0.29	18/25	0.86	14/51
0.33	3/109	1.00	0/12
0.43	15/18		
0.50	0/41		
0.57	19/56		

Table 2. Numbers of problems reduced by c-abstraction

(a) Domain

ALG	0/15	GRA	0/1	LCL	4/527	RNG	0/104
ANA	4/21	GRP	17/791	LDA	0/23	ROB	0/38
BOO	1/133	HAL	0/0	MGT	3/78	SET	5/706
CAT	2/62	HEN	0/67	MSC	2/13	SWC	0/423
COL	13/193	HWC	0/6	NLP	156/258	SWV	5/21
COM	0/8	HWV	31/83	NUM	5/315	SYN	462/839
FLD	0/279	KRS	0/17	PLA	2/32	TOP	0/24
GEO	6/253	LAT	1/104	PUZ	6/82		

(b) Ratio(%)

0-9	87	50-59	19
10-19	6	60-69	55
20-29	32	70-79	118
30-39	17	80-89	167
40-49	108	90-99	116

Table 2 (b) shows the ratio of remaining clauses to the original ones. For example, the first row indicates that there are 87 problems less than 10 % clauses of which are remaining after reduction. There are 57 problems which are reduced to the empty sets. All such problems are determined as satisfiable without proof.

In order to measure the c-abstraction effect, we solved all 725 problems, by using three provers with a time limit of 600 seconds: DCTP 1.31 [12], Vampire 7.0 [9], and E 0.82 [11][3]. These provers attended the CADE ATP system competition CASC-J2[16]. Vampire 7.0 won the first place in the MIX and FOF divisions, DCTP 1.31 won the third place in the EPR division, and E 0.82 won the third place in the MIX division.

Table 3 (a) shows summaries of c-abstraction effects on these three provers. The "before" column shows statistics for the original clause sets, while the "after" column shows statistics for their reduced clause sets. The last row shows the average cpu time in seconds. The parenthetic number in the "after" column shows the cpu time including preprocessing. Table 3 (b) shows a detailed version of (a). We succeed in enhancing performance of the three provers: The numbers of problems solved are increased from 642 to 647 for DCTP, from 626 to 642 for Vampire, and from 650 to 661 for E.

[3] DCTP runs with options "-negpref -complexity -fullrewrite -alternate -resisol".
Vampire runs with options "–mode casc-j2 -p off -t 600".
E runs with options "-s –print-statistics -xAuto -tAuto –memory-limit=384 –tptp-in".

Table 3. Effect of c-abstraction

(a) Summary

	DCTP 1.31		Vampire 7.0		E 0.82	
ALL	before	after	before	after	before	after
Attempted	725					
Solved	642	647	626	642	650	661
Av. Time(s)	6.15	5.10(9.28)	11.69	7.68(8.80)	5.48	11.64(15.73)

(b) Category summary

Domain	DCTP 1.31		Vampire 7.0		E 0.82	
(Attempted)	before	after	before	after	before	after
ANA	0	1	1	1	1	1
(4)	-	0.0(0.08)	15.53	15.28(15.36)	46.61	36.98(37.06)
BOO	1	1	1	1	1	1
(1)	0.0	0.0(0.07)	150.09	0.46(0.53)	22.66	0.50(0.57)
CAT	1	1	0	0	0	0
(2)	0.0	0.0(0.08)	-	-	-	-
COL	4	4	4	4	0	0
(13)	12.69	6.61(6.73)	159.39	159.22(159.33)	-	-
GEO	1	1	1	1	1	1
(6)	0.0	0.0(0.07)	0.73	0.11(0.18)	0.86	0.48(0.55)
GRP	17	17	7	8	9	11
(17)	1.05	1.05(1.12)	3.45	3.11(3.19)	1.62	1.35(1.43)
HWV	22	25	28	28	30	29
(31)	1.50	0.08(0.17)	12.76	4.67(4.77)	2.04	10.50(10.59)
LAT	1	1	0	0	0	0
(1)	0.01	0.01(0.09)	-	-	-	-
LCL	4	4	1	4	0	4
(4)	0.00	0.00(0.08)	235.93	0.37(0.45)	-	0.85(0.93)
MGT	3	3	3	3	3	3
(3)	0.03	0.03(0.11)	11.30	11.39(11.47)	0.51	0.52(0.60)
MSC	2	2	2	2	2	2
(2)	0.01	0.0(0.07)	0.41	0.27(0.34)	0.49	0.52(0.59)
NLP	126	126	140	140	146	146
(156)	0.05	0.04(0.14)	9.96	8.56(8.66)	3.19	33.17(33.27)
NUM	5	5	2	5	2	5
(5)	0.0	0.0(0.08)	0.12	0.33(0.40)	0.49	0.60(0.67)
PLA	2	2	2	2	2	2
(2)	0.01	0.0(0.08)	0.13	0.12(0.20)	0.49	0.48(0.56)
PUZ	6	6	5	6	6	6
(6)	0.03	0.03(0.11)	0.13	0.27(0.35)	0.51	0.51(0.59)
SET	4	4	4	4	3	4
(5)	0.09	0.05(0.13)	79.62	18.24(18.31)	0.49	82.65(82.73)
SWV	5	5	4	5	4	5
(5)	0.01	0.0(0.08)	84.62	0.31(0.39)	0.62	0.57(0.65)
SYN	438	439	421	428	440	441
(462)	8.76	7.39(13.52)	9.05	6.57(8.20)	6.69	4.87(10.95)

There is a series of 10 satisfiable problems in the NLP domain which have a negative effect on E. The average proving times for these problems are increased by 440 seconds after c-abstraction. This is a major cause why the average run time of E is increased from 5.48 seconds to 11.68 seconds.

C-abstraction effects on Vampire and E seem to be stronger than that on DCTP. This is due to the fact that there exist problems which have no positive clause or no negative clause. These problems are obviously satisfiable. But, for such problems, Vampire and E sometimes consume a lot of cpu time (or reach a time limit) while DCTP immediately stops. There are 18 such cases for Vampire and 11 cases for E. These problems become empty sets by c-abstraction, so it is easy to determine them as satisfiable[4]

Table 4. Positive and negative effects of c-abstraction

Problem	Time	No. of clauses		DCTP 1.31		Vampire 7.0		E 0.82	
	(secs)	original	reduced	before	after	before	after	before	after
ANA006-1	0.08	14	5	T.O.	0.00	T.O.	T.O.	T.O.	T.O.
BOO008-1	0.07	21	1	0.00	0.00	150.09	0.46	22.66	0.50
COL092-2	0.12	195	183	24.66	12.96	T.O.	T.O.	T.O.	T.O.
HWV009-1	0.10	92	66	T.O.	0.10	0.54	0.20	0.56	0.57
HWV031-1	0.10	93	67	T.O.	T.O.	T.O.	T.O.	35.22	283.71
NLP037-1	0.08	66	12	0.02	0.00	65.30	0.10	0.51	0.50
NLP186-1	0.11	108	99	T.O.	T.O.	0.25	0.26	34.69	538.11
NUM288-1	0.07	12	0	0.0	0.0	T.O.	0.44	T.O.	0.85
SET787-1	0.08	14	12	0.34	0.21	71.12	71.55	T.O.	329.16
SWV017-1	0.09	37	5	0.01	0.00	215.64	0.44	0.55	0.51
SYN597-1	0.08	28	23	5.36	5.37	30.91	209.20	3.47	T.O.
SYN599-1	0.08	29	25	89.48	89.57	209.29	162.53	42.30	5.07
SYN610-1	0.08	30	26	T.O.	T.O.	15.18	209.38	5.17	2.87
SYN624-1	0.07	35	26	0.25	0.25	4.80	61.30	111.65	0.68
SYN708-1	0.09	83	61	T.O.	T.O.	36.04	124.57	72.27	60.82
SYN742-1	0.08	31	0	0.04	0.00	169.80	0.10	0.67	0.49
SYN813-1	1.69	504	378	34.41	T.O.	13.59	9.19	10.71	2.96
SYN818-1	104.59	2621	1397	258.68	87.32	T.O.	T.O.	59.65	18.31
SYN821-1	27.53	1716	712	56.24	122.71	T.O.	26.52	T.O.	5.42
SYN822-1	33.15	1768	1138	65.15	34.17	T.O.	59.11	T.O.	71.08
SYN897-1	0.86	122	97	T.O.	4.85	2.26	1.87	1.35	1.11
SYN912-1	2.72	1780	247	61.90	0.62	T.O.	T.O.	T.O.	T.O.

Table 4 shows problems which exhibit positive or negative effect of c-abstraction on the 3 provers. The second column shows cpu times for preprocessing in seconds, the third the number of original clauses, and the fourth the number of

[4] Vampire regards an empty clause set as an error of the input and aborts. We treat such a case as a normal proof which tells that the set is satisfiable.

clauses remaining after preprocessing. The last 6 columns show proving time of the 3 provers. The "before" column shows the time for the original clause sets, while the "after" column shows the time for their reduced clause sets. "T.O." indicates that the problem is not solved in 600 seconds.

We succeeded in enhancing the provers' performance for several problems. For example on SYN912-1, DCTP's proving times is decreased from 61.90 seconds to 0.62 seconds after 2.72 seconds preprocessing which decreases the number of clauses from 1780 to 247. Vampire and E can prove SYN822-1 in 59.11 and 71.08 seconds respectively after preprocessing, while they cannot prove it within 600 seconds before preprocessing.

On the other hand, there are some problems which show negative effects of c-abstraction. For example on NLP186-1, E's proving time is increased from 34.69 seconds to 538.11 seconds. DCTP can not prove SYN813-1 within 600 seconds after preprocessing, while it can prove the same problem in 34.41 seconds before preprocessing. There is another type of problem which show both positive and negative effects. For example, SYN624-1 shows a negative effect on Vampire and a positive effect on E: Vampire's proving times is increased from 4.80 seconds to 61.30 seconds while E's proving time is decreased from 111.65 seconds to 0.68 seconds.

There are some satisfiable problems which are reduced to empty sets of clauses in DCTP's preprocessing phase after c-abstraction based clause elimination. In these problems, 48 problems are not reduced to empty sets without c-abstraction. This indicates that c-abstraction based clause elimination enhances the effects of other preprocessing operations.

4.3 C-Abstraction Based Elimination and Pure Literal Elimination

A pure literal is a literal in a clause that cannot be resolved against any literal in any clause. Clauses that contain pure literals can be eliminated, because such a clause cannot contribute a resolution proof. Pure literal elimination has a similar effect to c-abstraction's because c-abstraction based preprocessing eliminates clauses which contain literals irrelevant to model generation.

Pure literal elimination is sometimes stronger and sometimes weaker than c-abstraction based elimination. The strength comes from a unification operation which is necessary to the former but unnecessary to the latter. On the other hand, weakness comes from (model generation) inferences which are necessary to the latter but unnecessary to the former.

In 5522 CNF problem, 562 problems are reduced by pure literal elimination. This indicates that c-abstraction is applicable to more problems than pure literal elimination. The average elimination time is 4.49 seconds for 562 problems. More than 85% of the problems are reduced within one second for each, while 38 problems needs more than ten seconds for reducing. Pure literal elimination takes more time than c-abstraction does on average. This is caused by the task of unification.

Table 5 (a) shows the numbers of problems reduced by pure literal elimination for every problem domain. Table 5 (b) shows the ratio of remaining clauses to

Table 5. Pure literal elimination

(a) Domain

ALG	0/15	GRA	0/1	LCL	0/527	RNG	0/104
ANA	3/21	GRP	13/791	LDA	0/23	ROB	0/38
BOO	0/133	HAL	0/0	MGT	4/78	SET	2/706
CAT	1/62	HEN	0/67	MSC	0/13	SWC	0/423
COL	13/193	HWC	0/6	NLP	142/258	SWV	0/21
COM	2/8	HWV	6/83	NUM	0/315	SYN	358/839
FLD	0/279	KRS	0/17	PLA	1/32	TOP	0/24
GEO	6/253	LAT	0/104	PUZ	11/82		

(b) Ratio(%)

0-9	18	50-59	13
10-19	4	60-69	27
20-29	3	70-79	43
30-39	5	80-89	98
40-49	2	90-99	350

(c) Effect

ALL	DCTP 1.31		Vampire 7.0		E 0.82	
	before	after	before	after	before	after
Attempted	562					
Solved	509	510	485	485	507	509
Av. Time(s)	6.98	6.77(11.63)	7.10	6.70(8.25)	4.82	13.37(18.15)

the original ones. There are 350 problems which are in the ratio from 90% to 99%. This is 3 times as many as those of c-abstraction (cf. Table 2). We may say that the clause elimination effect of c-abstraction is generally stronger than that of pure literal elimination.

Table 5 (c) shows the summaries of pure literal elimination effects on the provers. A little effect can be seen on the performance of the provers. Indeed, there is no change in terms of problems solved within 600 seconds after preprocessing for Vampire. The influence of pure literal elimination upon the performance of these provers is weaker than that of c-abstraction.

There are 752 problems which are reduced by c-abstraction based elimination or pure literal elimination. They can be classified into 4 groups by the set inclusion relation as follows: (1) A is a proper subset of B, (2) A equals B, (3) A is a proper superset of B, and (4) there is no set inclusion relation between A and B, where A is a problem (i.e. a set of clauses) reduced by c-abstraction base elimination and B is a problem reduced by pure literal elimination. There are 333 problems in the first group, 182 in the second, 62 in the third, and 175 in the fourth. This indicates that pure literal elimination is different from c-abstraction. And it seems reasonable to suppose that the latter gains the ascendancy over the former with respect to clause elimination.

By the way, it is possible to apply pure literal elimination after c-abstraction. Our experiment shows that only simple problems are further reduced by pure literal elimination after c-abstraction. Thus, pure literal elimination after c-abstraction has no influence upon the performance of DCTP, Vampire, and E.

4.4 C-Abstraction Combined with D-Abstraction

Among 725 problems reduced by c-abstraction, there are 87 problems which are determined as satisfiable by d-abstraction. We don't need to prove these problems anymore. Is is natural to combine c-abstraction with d-abstraction. Table 6 shows the summary of c-abstraction effects on the remaining 638 problems. The parenthetic number in the "after" column shows the cpu time including c-abstraction and d-abstraction.

Table 6. Effect of C-abstraction on problems passed through d-abstraction

	DCTP 1.31		Vampire 7.0		E 0.82	
ALL	before	after	before	after	before	after
Attempted	638					
Solved	556	560	565	567	585	583
Av. Time(s)	7.06	5.86(10.83)	8.86	8.71(10.09)	5.97	13.10(17.87)

There is a positive effect on DCTP. The number of problems solved is increased from 556 to 560 and the average cpu time is decreased from 7.06 seconds to 5.86 seconds. For Vampire, c-abstraction barely has a positive effect. The number of problems solved is increased from 565 to 567, but the average cpu time is almost unchaged. Unfortunately, there is a negative effect on E. The number of problems solved is decreased from 585 to 583 and the average cpu time is increased from 5.97 seconds to 13.10 seconds.

5 Conclusion

Preprocessing a set of clauses has a great impact on the success of a subsequent automated reasoning system. We have introduced two abstractions of the given clause set for preprocessing it. Experimental results show that these abstractions are effective for several problems. 29% of satisfiable first-order problems in TPTP are determined as satisfiable with their d-abstract clause sets. 13% of CNF problems in TPTP are reduced with d-abstraction.

C-abstraction sometimes has positive effects and sometimes negative effects on state-of-the-art theorem provers: DCTP, Vampire, and E. As a whole, without d-abstraction, these provers profit from c-abstraction. On the other hand for the problems passed through d-abstraction, DCTP and Vampire profit from c-abstraction, but E does not. This situation may be improved if we find a combination of the provers' options that fit for c-abstraction. Furthermore, the combination of the proposed method and other preprocessing operations can enhance their abilities.

Acknowledgement

This research was partially supported by Grant-in-Aid for Scientific Research (No. A-15200002) from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. François Bry and Adnan Yahya. Positive Unit Hyperresolution Tableaux and Their Application to Minimal Model Generation. *Journal of Automated Reasoning*, 25:35–82, 2000.
2. Marc Fuchs and Dirk Fuchs. Abstraction-Based Relevancy Testing for Model Elimination. In Harald Ganzinger, editor, *CADE-16*, volume 1632 of *LNAI*, pages 344–358. Springer, July 1999.
3. Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57:323–389, 1992.
4. Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing Abstraction Hierarchies for Planing. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 692–697. AAAI Press, 1991.
5. Miyuki Koshimura and Ryuzo Hasegawa. Proof Simplification for Model Generation and Its Applications. In Michel Parigot and Andrei Voronkov, editors, *LPAR2000*, volume 1955 of *LNAI*, pages 96–113. Springer, November 2000.
6. Rainer Manthey and François Bry. SATCHMO: a theorem prover implemented in Prolog. In E. Lusk and R. Overbeek, editors, *CADE-9*, volume 310 of *LNCS*, pages 415–434. Springer, May 1988.
7. David A. Plaisted. Theorem Proving with Abstraction. *Artificial Intelligence*, 16:47–108, 1981.
8. David A. Plaisted and Adnan Yahya. A relevance restriction strategy for automated deduction. *Artificial Intelligence*, 144:59–93, 2003.
9. Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2):91–110, 2002.
10. Earl D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.
11. Stephan Schulz. E - a brainiac theorem prover. *AI Communications*, 15(2):111–126, 2002.
12. Gernot Stenz. DCTP 1.2 - System Abstract. In Uwe Egly and Christian G. Fernmuller, editors, *TABLEAUX-2002*, volume 2381 of *LNAI*, pages 335–340. Springer, 2002.
13. G. Sutcliffe and C.B. Suttner. The TPTP Problem Library: CNF Release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.
14. G. Sutcliffe and C.B. Suttner. The TPTP Problem Library for Automated Theorem Proving. `http://www.cs.miami.edu/~tptp/`, 2004.
15. Geoff Sutcliffe and Stuart Melville. The Practice of Clausification in Automatic Theorem Proving. *South African Computer Journal*, 18:57–68, 1996.
16. Geoff Sutcliffe and Christian Suttner. The CADE ATP System Competition. In David Basin and Michaël Rusinowitch, editors, *IJCAR 2004*, volume 3097 of *LNAI*, pages 490–491. Springer, July 2004.
17. Neng-Fa Zhou. *B-Prolog User's Manual (Version 6.6).* http://www.probp.com, 2004.