System Description of a SAT-based CSP Solver Sugar

Naoyuki Tamura[1], Tomoya Tanjo[2], and Mutsunori Banbara[1]

[1] Information Science and Technology Center, Kobe University, JAPAN
{tamura,banbara}@kobe-u.ac.jp
[2] Graduate School of Engineering, Kobe University, JAPAN

Abstract. This paper gives the system description of a SAT-based CSP solver Sugar submitted to the Third International CSP Solver Competition. The Sugar solver solves a finite linear CSP and MAX-CSP by translating it into a SAT problem using the order encoding method and then solving the translated SAT problem with an external SAT solver (e.g. MiniSat). In the order encoding method, a comparison $x \le a$ is encoded by a different Boolean variable for each integer variable x and integer value a.

1 Introduction

This paper gives the system description of a SAT-based CSP solver *Sugar* [3] submitted to the Third International CSP Solver Competition.

The Sugar solver solves a finite linear CSP and MAX-CSP by translating it into a SAT problem by using *order encoding* method [1, 2] and then solving the translated SAT problem by a SAT solver, such as MiniSat [3] and PicoSAT [4].

The method of the order encoding is basically the same with the one used for job-shop scheduling problems by Crawford and Baker in [5] and studied by Soh, Inoue, and Nabeshima in [6–8]. It encodes a comparison $x \le a$ by a different Boolean variable for each integer variable x and integer value a.

The benefit of this encoding is the natural representation of the order relation on integers compared with the other encoding methods, such as direct encoding and support encoding, used by other SAT-based CSP solvers [9–13].

Axiom clauses with two literals, such as $\{\neg(x \le a), x \le a+1\}$ for each integer a, represent the order relation of an integer variable x. Clauses, for example $\{x \le a, \neg(y \le a)\}$ for each integer a, can be used to represent the constraint among integer variables, i.e. $x \le y$.

2 Implementation

This section describes some implementation details of the Sugar solver.

[3] http://bach.istc.kobe-u.ac.jp/sugar/

```
(domain D0 0 2)
(int V0 D0)
(int V1 D0)
(predicate (P0 X0 X1) (ne X0 X1))
(relation R0 2 (conflicts (0 0) (1 1) (2 2)))
(P0 V0 V1)
(R0 V0 V1)
(alldifferent (V0 V1))
```

Fig. 1. Example of Sugar CSP description

2.1 Preprocessing

At a preprocessing stage, XCSP 2.1 file in the abridged notation is translated into a Sugar CSP file written in a Lisp-like list format.

Fig. 1 shows an example of the Sugar CSP file.

2.2 Conversion to a Clausal Form CSP

Before encoding CSP to SAT, Sugar converts the CSP into its clausal form (that is, Conjunctive Normal Form).

A clausal form CSP consists of a set of CSP literals, and a CSP literal is one of the followings:[4]

- A Boolean literal: p or $\neg p$ where p is a Boolean variable.
- A linear comparison literal: $\sum a_i x_i \le b$ where a_i's and b are integer constants and x_i's are integer variables.
- A relation literal: a set of conflict tuples or support tuples with integer variable arguments (for example, `(R0 V0 V1)` in the Fig. 1 is represented as a relation literal).

Expressions other than $\sum a_i x_i \le b$ are translated by using the conversion rules described in the Fig. 2 where E div c and E mod c are integer quotient and remainder of E divided by an integer constant c respectively.

Basically, Sugar is not able to handle non-linear expressions in the current implementation except some special cases. For example, $xy < 0$ is translated into $((x < 0) \land (y > 0)) \lor ((x > 0) \land (y < 0))$, and xy is translated into $\mathrm{if}(x = a_1,\ a_1 y,\ a_2 y)$ when the domain of x is $\{a_1, a_2\}$.

The $\mathrm{alldifferent}(x_1, x_2, \ldots, x_n)$ constraint is translated into $\bigwedge_{i<j}(x_i \ne x_j)$ with extra pigeon hole constraints $\neg \bigwedge(x_i < lb+n-1)$ and $\neg \bigwedge(x_i > ub-n+1)$ where lb and ub are the lower and upper bounds of $\{x_1, x_2, \ldots, x_n\}$. Other global constraints are translated in a straightforward way.

Finally, constraints are converted into clausal form by using the Tseitin transformation of introducing new Boolean variables.

[4] Literals for multiplications and power functions will be used in a future implementation.

Expression	Replacement	Extra condition
$E < F$	$E + 1 \le F$	
$E = F$	$(E \le F) \wedge (E \ge F)$	
$E \ne F$	$(E < F) \vee (E > F)$	
$\max(E, F)$	x	$(x \ge E) \wedge (x \ge F) \wedge ((x \le E) \vee (x \le F))$
$\min(E, F)$	x	$(x \le E) \wedge (x \le F) \wedge ((x \ge E) \vee (x \ge F))$
$\mathrm{abs}(E)$	x	$(x \ge E) \wedge (x \ge -E) \wedge ((x \le E) \vee (x \le -E))$
E div c	q	$(E = c\,q + r) \wedge (0 \le r) \wedge (r < c)$
E mod c	r	$(E = c\,q + r) \wedge (0 \le r) \wedge (r < c)$
$\mathrm{if}(C, E, F)$	x	$(C \supset x = E) \wedge (\neg C \supset x = F)$

Fig. 2. Encoding expressions other than $\sum a_i x_i \le b$

2.3 Constraint Propagation

Constraint propagation is done for the clausal form CSP to reduce the redundant integer variable values, CSP clauses, and CSP literals. AC-3 algorithm is used in the current implementation.

For example, more than half billion values were removed in the FISCHER11-6-fair instance.

2.4 SAT encoding

The clausal form CSP is encoded into a SAT problem by using the order encoding method [1].

Compared with the previous version submitted to the Second CSP solver competition [2], conflict tuples in extensional constraints are combined into conflict regions. For example, conflict tuples $((0, 1), (0, 2))$ for variables x and y was encoded into two clauses $\neg((x \ge 0) \wedge (x \le 0) \wedge (y \ge 1) \wedge (y \le 1))$ and $\neg((x \ge 0) \wedge (x \le 0) \wedge (y \ge 2) \wedge (y \le 2))$ in the previous version. They are now encoded into one clause $\neg((x \ge 0) \wedge (x \le 0) \wedge (y \ge 1) \wedge (y \le 2))$ by combining into coflict regions.

Support tuples are encoded by considering their complement space.

2.5 SAT solver

Sugar can use MiniSat [3] or PicoSAT [4] as an external SAT solver.

MiniSat is well known to be very efficient especially for unsatisfiable problems and widely used in many application areas. PicoSAT is a solver based on MiniSat with rapid restarts and reusing phases of assigned variables.

In our experiments, MiniSat is slightly better for many problems, PicoSAT, however, uses less memory and can solve some problems which are not solved by MiniSat under the CSP solver competition environment (that is, 900MiB memory limitation).

3 Conclusion

In this paper, we have described some implementation details of Sugar constraint solver submitted to the Third International CSP Solver Competition. The Sugar solver solves a finite linear CSP and MAX-CSP by translating it into a SAT problem using the order encoding method and then solving the translated SAT problem with an external SAT solver (e.g. MiniSat).

Acknowledgments

We would like to give thanks to the competition organizers for their efforts and Katsumi Inoue, Hidetomo Nabeshima, Takehide Soh, and Shuji Ohnishi for their helpful suggestions.

References

1. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. In: Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006). (2006) 590–603
2. Tamura, N., Banbara, M.: Sugar: A CSP to SAT translator based on order encoding. In: Proceedings of the Second International CSP Solver Competition. (2008) 65–69
3. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003). (2003) 502–518
4. Biere, A.: Adaptive restart strategies for conflict driven SAT solvers. In: Proceedings of the Theory and Applications of Satisfiability Testing (SAT 2008). (2008) 28–33
5. Crawford, J.M., Baker, A.B.: Experimental results on the application of satisfiability algorithms to scheduling problems. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94). (1994) 1092–1097
6. Soh, T., Inoue, K., Banbara, M., Tamura, N.: Experimental results for solving job-shop scheduling problems with multiple SAT solvers. In: Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing (DSCP'05). (2005)
7. Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M., Tamura, N.: A competitive and cooperative approach to propositional satisfiability. Discrete Applied Mathematics **154** (2006) 2291–2306
8. Nabeshima, H., Soh, T., Inoue, K., Iwanuma, K.: Lemma reusing for SAT based planning and scheduling. In: Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS'06). (2006) 103–112
9. Biere, A.: Translating CSP to SMV and then to SAT for the CSP'05 competition. In: Proceedings of the Second International Workshop on Constraint Propagation and Implementation, Volume II Solver Competition. (2005) 49–52
10. Berre, D.L.: CSP2SAT4J: A simple CSP to SAT translator. In: Proceedings of the Second International Workshop on Constraint Propagation and Implementation, Volume II Solver Competition. (2005) 59–66

11. Roussel, O.: Some notes on the implementation of csp2sat+zchaff, a simple translator from CSP to SAT. In: Proceedings of the Second International Workshop on Constraint Propagation and Implementation, Volume II Solver Competition. (2005) 83–88
12. van Dongen, M.R.C., Lecoutre, C., Roussel, O.: Results of the second CSP solver competition. In: Proceedings of the Second International CSP Solver Competition. (2008) 1–10
13. Berre, D.L., Lynce, I.: CSP2SAT4J: A simple CSP to SAT translator. In: Proceedings of the Second International CSP Solver Competition. (2008) 43–54