Sugar: A CSP to SAT Translator Based on Order Encoding

Naoyuki Tamura and Mutsunori Banbara

Information Science and Technology Center, Kobe University, JAPAN
{tamura,banbara}@kobe-u.ac.jp

Abstract. This paper gives some details on the implementation of *sugar* constraint solver submitted to the Second International CSP Solver Competition. The sugar solver solves a finite linear CSP by translating it into a SAT problem by using *order encoding* method and then solving the translated SAT problem by the MiniSat solver. 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 some details on the implementation of *sugar* constraint solver submitted to the Second International CSP Solver Competition.

The sugar solver solves a finite linear CSP by translating it into a SAT problem by using *order encoding* method [1] and then solving the translated SAT problem by the MiniSat solver [2].

The method of the order encoding is basically the same with the one used for Job-Shop Scheduling Problems by Crawford and Baker in [3] and studied by Soh, Inoue, and Nabeshima in [4–6]. 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. 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 Order encoding

The order encoding uses Boolean variables p_{xi} meaning $x \le i$ for each CSP variable x and each integer constant i ($\ell(x) - 1 \le i \le u(x)$) where $\ell(x)$ and $u(x)$ are the lower and upper bounds of x respectively[1].

[1] $p_{x\ell(x)-1}$ and $p_{xu(x)}$ are redundant because they are always false and true respectively. However, we use them for simplicity of the discussion.

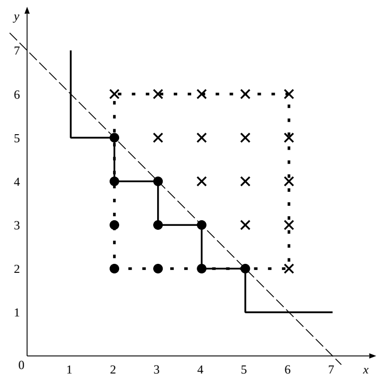

Fig. 1. Encoding $x + y \le 7$

Consider an example of encoding $x + y \le 7$ when $x, y \in \{2, 3, 4, 5, 6\}$. The following 12 Boolean variables are used to encode the example.

$$\begin{array}{cccccc} p_{x1} & p_{x2} & p_{x3} & p_{x4} & p_{x5} & p_{x6} \\ p_{y1} & p_{y2} & p_{y3} & p_{y4} & p_{y5} & p_{y6} \end{array}$$

The following clauses are used as axioms representing the bounds and the order relation for each CSP variable x.

$$\begin{array}{ll} \neg p_{x\,\ell(x)-1} & \\ p_{x u(x)} & \\ \neg p_{x\,i-1} \lor p_{xi} & (\ell(x) \le i \le u(x)) \end{array}$$

Therefore, the following 14 clauses are required for the example.

$$\begin{array}{ccccc} & \neg p_{x1} & & p_{x6} & \\ \neg p_{x1} \lor p_{x2} & \neg p_{x2} \lor p_{x3} & \neg p_{x3} \lor p_{x4} & \neg p_{x4} \lor p_{x5} & \neg p_{x5} \lor p_{x6} \end{array}$$
$$\text{(similar clauses for } y\text{)}$$

Constraints are encoded into clauses representing conflict regions instead of conflict points. When all points $(x_1, \ldots, x_n)$ in the region $i_1 < x_1 \le j_1, \;\ldots,\; i_n < x_n \le j_n$ violate the constraint, the following clause is added.

$$p_{x_1 i_1} \lor \neg p_{x_1 j_1} \lor \cdots \lor p_{x_n i_n} \lor \neg p_{x_n j_n}$$

Therefore, the following 5 clauses are used to encode $x + y \le 7$ (Fig.1).

$$p_{x1} \lor p_{y5} \qquad p_{x2} \lor p_{y4} \qquad p_{x3} \lor p_{y3} \qquad p_{x4} \lor p_{y2} \qquad p_{x5} \lor p_{y1}$$

When a_i's are non-zero integer constants, c is an integer constant, and x_i's are mutually distinct integer variables, any finite linear comparison $\sum_{i=1}^{n} a_i\, x_i \le c$

can be encoded into the following CNF formula [1].

$$\bigwedge_{\sum_{i=1}^{n} b_i = c-n+1} \bigvee_{i} (a_i\, x_i \le b_i)^{\#}$$

Parameters b_i's range over integers satisfying $\sum_{i=1}^{n} b_i = c-n+1$ and $\ell(a_i x_i)-1 \le b_i \le u(a_i x_i)$ for all i where functions ℓ and u give the lower and upper bounds of the given expression respectively. The translation $()^{\#}$ is defined as follows.

$$(a\,x \le b)^{\#} \;\equiv\; \begin{cases} x \le \left\lfloor \dfrac{b}{a} \right\rfloor & (a > 0) \\ \neg\left(x \le \left\lceil \dfrac{b}{a} \right\rceil - 1\right) & (a < 0) \end{cases}$$

3 System Description of Sugar

Sugar is a CSP to SAT solver based on the order encoding. It consists of the front-end Perl program and the encoder program written in Java[2]. The MiniSat 1.4 [2] is used as the backend SAT solver in the submitted version.

CSP instances are encoded into SAT instances in the following ways.

Encoding m-ary linear comparisons: As described in the previous section, comparisons of the form $\sum_{i=1}^{m} a_i x_i \le b$ can be encoded into $O(d^{m-1})$ clauses in general where d is the domain size.

However, it is possible to reduce the number of integer variables in each comparison at most three by introducing new integer variables. Therefore, each comparison $\sum_{i=1}^{m} a_i\, x_i \le b$ can be encoded by at most $O(m\, d^2)$ clauses even when $m \ge 4$.

Encoding other expressions: Expressions other than $\sum a_i x_i \le b$ are encoded to SAT formulas by using the conversion described in the Fig.2 where E div c and $E \bmod c$ are integer quotient and remainder of E divided by an integer constant c.

Expression at the first column is replaced with the replacement at the second column with some extra condition at the third column.

Note that non-linear expressions such as $x \times y$ can not be handled by the sugar program submitted to the competition.

Keeping clausal form: When encoding a clause of CSP to SAT, the encoded formula is no more a clausal form in general. As it is well known, introduction of new Boolean variables is useful to solve this problem.

[2] The package is available at `http://bach.istc.kobe-u.ac.jp/sugar/`

Expression	Replacement	Extra condition
$E < F$	$E + 1 \le F$	
$E = F$	$(E \le F) \land (E \ge F)$	
$E \ne F$	$(E < F) \lor (E > F)$	
$\max(E, F)$	x	$(x \ge E) \land (x \ge F) \land ((x \le E) \lor (x \le F))$
$\min(E, F)$	x	$(x \le E) \land (x \le F) \land ((x \ge E) \lor (x \ge F))$
$\text{abs}(E)$	$\max(E, -E)$	
E div c	q	$(E = c\,q + r) \land (0 \le r) \land (r < c)$
E mod c	r	$(E = c\,q + r) \land (0 \le r) \land (r < c)$

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

Consider an example of encoding a clause $\{x - y \le -1, -x + y \le -1\}$ when $x, y \in \{0, 1, 2\}$. Comparisons $x - y \le -1$ and $-x + y \le -1$ are converted into $S_1 = (x \le -1 \lor \neg(y \le 0)) \land (x \le 0 \lor \neg(y \le 1)) \land (x \le 1 \lor \neg(y \le 2))$ and $S_2 = (\neg(x \le 2) \lor y \le 1) \land (\neg(x \le 1) \lor y \le 0) \land (\neg(x \le 0) \lor y \le -1)$ respectively. Expanding $S_1 \lor S_2$ generates 9 clauses. However, by introducing new Boolean variables p and q, we obtain the following seven clauses.

$$\begin{array}{ccc} & \{p, q\} & \\ \{\neg p, x \le -1, \neg(y \le 0)\} & \{\neg p, x \le 0, \neg(y \le 1)\} & \{\neg p, x \le 1, \neg(y \le 2)\} \\ \{\neg q, \neg(x \le 2), y \le 1\} & \{\neg q, \neg(x \le 1), y \le 0\} & \{\neg q, \neg(x \le 0), y \le -1\} \end{array}$$

Encoding extensional constraints: Extensional constraints with conflict tuples and support tuples are encoded by a simple way in the submitted version of sugar.

Conflict tuples $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ for variables (x, y) are encoded as follows.

$$\neg(x = a_1 \land y = b_1) \land \cdots \land \neg(x = a_n \land y = b_n)$$

Support tuples $\{(a_1, b_1), \ldots, (a_n, b_n)\}$ for variables (x, y) are encoded as follows.

$$(x = a_1 \land y = b_1) \lor \cdots \lor (x = a_n \land y = b_n)$$

4 Conclusion

In this paper, we have described some details on the implementation of sugar constraint solver submitted to the Second International CSP Solver Competition. The sugar solver solves a finite linear CSP by translating it into a SAT problem by using order encoding method and then solving the translated SAT problem by the MiniSat solver. Although the system is still under development, we hope it gives some research directions for CSP to SAT encoding systems.

Acknowledgments

We would like to give thanks to 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. 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
3. 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
4. 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)
5. Inoue, K., Soh, T., Ueda, S., Sasaura, Y., Banbara, M., Tamura, N.: A competitive and cooperative approach to propositional satisfiability. Discrete Applied Mathematics (2006) (to appear).
6. 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