From Set Constraint Models to SAT Instances

Frédéric Lardeux, Eric Monfroy

From Set Constraint Models to SAT Instances

Frédéric Lardeux

LERIA - Université d'Angers.
Angers, France.
Email: Frederic.Lardeux@univ-angers.fr

Eric Monfroy

LINA - UMR 6241. TASC - INRIA
Université de Nantes, France.
Email: Eric.Monfroy@univ-nantes.fr

***Abstract*—On the one hand, Constraint Satisfaction Problems (CSP) are a declarative and expressive approach for modeling problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances up to millions of variables and clauses. In this article, we present an approach for taking advantage of both CSP modeling and SAT solving. Our technique consists in expressively modeling set constraint problems as CSPs that are automatically treated by some reduction rules to remove values that do not participate in any solution. These reduced CSPs are then encoded into "good" SAT instances that can be solved by standard SAT solvers. We illustrate our technique on the Sports Tournament Scheduling problem, and we show that we obtain competitive results compared to an ad-hoc solver. Our technique is simpler, more expressive, and less error-prone than direct SAT modeling. The SAT instances that we automatically generate are rather small and can efficiently be solved up to huge instances. Moreover, the reduction phase enables to push back the limits and treat even larger problems.**

I. Introduction

A classical way to formulate combinatorial problems is to use Constraint Satisfaction Problem (CSP) formulation [1]. A CSP is defined by some variables and constraints between these variables. Solving a CSP consists in finding assignments of the variables that satisfy the constraints. Expressiveness is one of the main strength of CSP: variables can be of various types (finite domains, floating point numbers, sets, ...) and constraints as well (linear arithmetic constraints, set constraints, non linear constraints, Boolean constraints, ...). Moreover, the so-called global constraints not only improve solving efficiency but also expressiveness: they propose new constructs and relations such as *alldifferent* (to enforce that all the variables of a list have different values), *cardinality* (to link a set to its size), ...

Another way to formulate combinatorial problems is to use the propositional satisfiability problem (SAT) [2]. A SAT problem is a Boolean formula written in conjunctive normal form, i.e., a conjunction of clauses. Clauses are disjunctions of literals. A literal is a variable or a negated variable. When all the clauses can be satisfied, the problem is said to be satisfiable. SAT is then restricted (in terms of expressiveness) to Boolean variables and propositional formulae. Coding set constraints directly into SAT is a tedious tasks (see for example [3] or [4]). Moreover, when one wants to optimize its model in terms of variables and clauses this quickly leads to very complicated and unreadable models in which errors can easily appear. However, SAT solvers can now handle huge SAT instances (millions of variables).

It is thus attractive to 1) encode CSPs into SAT (e.g., [5], [6], [7]) in order to benefit from the expressiveness of CSP and the power of SAT, and 2) introduce more expressiveness into SAT, e.g., with global constraints such as alldifferent [8], or cardinality [9].

Various systems of set constraints (either specialized systems [10], libraries for constraint programming systems such as [11], or the set constraint library of CHOCO [12] have been designed and it has been shown that numerous problems can easily be modeled with set constraints.

In this paper we are concerned with set constraint modeling, set constraint filtering, and their transformations into SAT instances: we often refer to this transformation as "encoding". Our goal is not to compete with standard CSP set solvers, but to introduce set constraints into SAT. However, we obtained very good results, and for problems such as the Sports Tournament Scheduling problem we are competitive with specialy designed solvers (see Section V).

In [13], we presented encoding rules that are directly applied on the CSP models. However, we have noticed that the set variables are not always as small as they could be: some elements could be removed without loosing any solution. Thus, the generated SAT instances could also be reduced. In [14], we introduced a constraint propagation phase to filter set variables. This filtering was only applied on upper bounds (elements eventually in the set) of set variables. The idea was to simplify the modeling phase by allowing not to specify precisely upper bounds of sets. Although interesting, this filtering was not strong enough to crucially reduce generated SAT instances. Furthermore, this representation was inefficient for propagating eventual assignments such as they appear when breaking symmetries.

In this article, we thus propose:

- a simple but complete and expressive set language for easily modeling problems with constraints such as intersection, union, cardinality, min, ... Compared to [13], [14], the language as been extended with finite domain variables, and comparison constraints between these variables. Moreover, cardinality is now a constraint linking a finite domain variable to a set variable. Similarly, we introduced constraints for minimum (maximum) of

sets. These new constraints are very useful for breaking symmetries.
- a set of reduction rules ($\Rightarrow_{red}$) to reduce CSP models. The fixed point application of these rules define a propagation algorithm, each rule corresponding to a filtering function ([15]) for sets and elements. The filtering is based on both bounds of sets and their cardinalities as defined in [16]. This filtering is much stronger than our previous one [14], stronger than bound consistency for sets [11], and weaker than [17].
- a set of encoding rules ($\Leftrightarrow_{enc}$) that convert CSP constraints into SAT instances. Compared to [14], we have more rules in order to cover the new constraints. Moreover our new rules can now be applied on reduced or not reduced constraints without generating unuseful clauses. Indeed, for each set appearing in a constraint, for each element, 3 cases of membership are considered: in the lower bound (the element is effectively in the set), in the upper bound but not in the lower bound (the element is eventually in the set), not in the upper bound (the element is not in the set). Thus, for a ternary constraint (such as $A = B \cap C$) this means 27 cases. After reduction, some of these cases vanish, and thus, no clause is generated.

We have successfully applied our technique on various problems (such as the Social Golfer Problem, the Sudoku puzzle, the n-queen problem, car sequencing, WhoWithWhom puzzle, ...) and we illustrate this paper with the Sports Tournament Scheduling problem (STS) [18]. The SAT instances which are automatically generated have a complexity similar to the complexity of improved directly-written SAT formulations. They are much smaller than the instances we could generate before and we can now tackle larger problems. Moreover, their solving with a SAT solver (in our case Minisat [19]) is efficient compared to other SAT approaches. For the STS problem, our approach is competitive with the solver of [20]. To our knowledge, this solver is the best one for the STS problem. Indeed, it has been especially designed and over constraints the STS to solve it. Thus, it may also be enable to find solution for some instances.

In the following section (Section II) we present our CSP set constraint language and the notions of upper/lower bounds, domains, etc. In Section III we show the rules to reduce models by eliminating elements that cannot participate in any solution. Section IV presents our new rule-based system for encoding set constraints into SAT. Section V illustrates our approach on the Sports Tournament Scheduling problem. We finally compare our work with existing ones in Section VI and conclude in Section VII.

II. Constraint Satisfaction Problems with Sets

A. Set CSP

Definition 1 (Set-CSP): A Set-CSP is defined by

- a universe $\mathcal{U}$ of integers.
- a set $X = \{x_1, \ldots, x_n\}$ of finite domain variables such that a domain of possible values $D_{x_i} \subseteq \mathcal{U}$ is associated to each variable $x_i \in X$. $\mathcal{D}$ denotes the Cartesian product $D_{x_1} \times \ldots \times D_{x_n}$
- a set $\mathbb{F}$ of set variables, such that each set variable $F \in \mathbb{F}$ is associated to:
  - its greatest lower bound $F^{\downarrow} \subseteq \mathcal{U}$
  - its lowest upper bound $F^{\uparrow} \subseteq \mathcal{U}$
  - its minimum cardinality $\#F^{\downarrow} \in \mathbb{N}$
  - its maximum cardinality $\#F^{\uparrow} \in \mathbb{N}$

  We also have that $F^{\downarrow} \subseteq F \subseteq F^{\uparrow}$, $\#F^{\downarrow} \leq |F| \leq \#F^{\uparrow}$, $|F^{\downarrow}| \leq \#F^{\downarrow}$, and $\#F^{\uparrow} \leq |F^{\uparrow}|$ where $|S|$ denotes the cardinality of the set S.
- a set of constraints C that are relations defined over $\mathcal{D}^{|X|} \times \mathcal{U}^{|\mathbb{F}|}$.

Note that $F^{\downarrow}$ (the greatest lower bound of a set $F \in \mathbb{F}$) represents integers that are required in F; $F^{\uparrow}$ the (lowest upper bound of F) represents integers that are eventually in F. Decomposing $F^{\uparrow}$, we denote by $F^{?}$ the set $F^{\uparrow} \setminus F^{\downarrow}$: elements of $F^{?}$ are eventually in F while elements of $F^{\uparrow} \setminus F^{?} = F^{\downarrow}$ are required in F. Thus, $F^{\downarrow} \subseteq F \subseteq F^{\uparrow}$ and possible values of F are elements of the powerset $2^{F^{\uparrow}}$ that includes $F^{\downarrow}$ and such that their cardinality is in the range $[\#F^{\downarrow}..\#F^{\uparrow}]$.

B. Basic Set Constraints

Variables are declared and initialized with the following constructs:

- the universe $\mathcal{U}$ is declared by $\mathcal{U} :: \top$ where $\top$ is a set of elements;
- an element variable x is declared by $x :: D_x$ where its domain D_x is given as a set of elements;
- a set variable F is declared by $F :: (F^{\downarrow}, F^{\uparrow}, \#F^{\downarrow}, \#F^{\uparrow})$ where its lower bound $F^{\downarrow}$ and its upper bound $F^{\uparrow}$ are sets of elements, and its minimum cardinality $\#F^{\downarrow}$, and its maximum cardinality $\#F^{\uparrow}$ are given by integers.

By abuse of notation, the empty set variable is denoted by $\emptyset$ and defined by $\emptyset :: (\emptyset, \emptyset, 0, 0)$.

Consider F, G, H, and F_i (i ranging from 1 to n) being set variables, and x being a finite domain variable. We enumerate here some usual (CSP) set constraints that we have considered:

finite domain (dis)equality	$x = y$	$(x \neq y)$
finite domain (strict) inequality	$x < y$	$(x \leq y)$
(non)membership	$x \in F$	$(x \notin F)$
set (dis)equality	$F = G$	$(F \neq G)$
inclusion	$F \subseteq G$	$(F \not\subseteq G)$
difference	$H = F \setminus G$	
intersection	$F = \bigcap_{i=1}^{n} F_i$	
union	$F = \bigcup_{i=1}^{n} F_i$	
partition	$F = \bigsqcup_{i=1}^{n} F_i$	
set cardinality	$x = \|F\|$	
set variable minimum	$x = min(F)$	
set variable maximum	$x = max(F)$	

The partition $F = \bigsqcup_{i=1}^{n} F_i$ means that $F = \bigcup_{i=1}^{n} F_i$, and for all i, j, $F_i \cap F_j = \emptyset$.

III. Reduction Rules

The $\Rightarrow_{red}$ reduction rules aim at reducing the search space. A fixed point application of these rules implement a propagation and filtering algorithm for sets and finite domains. The $\Rightarrow_{red}$ reduction rules may add elements to lower bounds of sets, remove elements from upper bounds of sets, increase minimum cardinality of sets, and decrease maximum cardinality of sets. For finite domain variable, they can only remove elements from domains. Moreover, some rules may also lead to a failure case, or remove some constraints that became unnecessary.

Our filtering algorithm enforces bound consistency with cardinality as defined in [10], [16].

In the following, we give some of our $\Rightarrow_{red}$ reduction rules to illustrate their use. More rules for bound consistency with cardinality are given in [10].

A. Elements

If a variable x has an empty domain, the CSP does not have any solution:

$$D_x = \emptyset \;\Rightarrow_{red}\; fail \tag{1}$$

When a variable x has been twice declared, the 2 declarations are grouped into one:

$$x :: D_x, x :: D'_x \;\equiv_{red}\; x :: D_x \cap D'_x \tag{2}$$

Note that applying Rule 2 replaces $x :: D_x$ and $x :: D'_x$ by $x :: D_x \cap D'_x$.

B. Sets

Rule 4 is very important: in some other rules, we can add elements to $F^{\downarrow}$ that may not be in $F^{\uparrow}$; Rule 4 will lead to a failure in these cases. Rules 4, 5 and 6 are useful when a cardinality constraint modifies the upper or lower bound of the cardinality of a set.

$$\#F^{\downarrow} > \#F^{\uparrow} \;\Rightarrow_{red}\; fail \tag{3}$$
$$F^{\downarrow} \not\subseteq F^{\uparrow} \;\Rightarrow_{red}\; fail \tag{4}$$
$$|F^{\downarrow}| > \#F^{\uparrow} \;\Rightarrow_{red}\; fail \tag{5}$$
$$|F^{\uparrow}| < \#F^{\downarrow} \;\Rightarrow_{red}\; fail \tag{6}$$

If $\#F^{\uparrow} = 0$ then F is the empty set (if $F^{\downarrow} \neq \emptyset$ this will lead to a failure with Rule 4):

$$\#F^{\uparrow} = 0, \;\Rightarrow_{red}\; F^{\uparrow} = \emptyset \tag{7}$$

Rules 8 and 9 make the set F to be a constant when the size of the upper bound is equal to the minimum cardinality of the set or when the size of the lower bound is equal to the maximum cardinality of the set:

$$\begin{array}{c} \#F^{\downarrow} = |F^{\uparrow}|, \;\; F^{\downarrow} \subset F^{\uparrow}, \;\; \#F^{\downarrow} \leq \#F^{\uparrow} \\ \Rightarrow_{red} \\ F^{\downarrow} \leftarrow F^{\uparrow}, \;\; \#F^{\uparrow} \leftarrow \#F^{\downarrow} \end{array} \tag{8}$$

$$\begin{array}{c} \#F^{\uparrow} = |F^{\downarrow}|, \;\; F^{\downarrow} \subset F^{\uparrow}, \;\; \#F^{\downarrow} \leq \#F^{\uparrow} \\ \Rightarrow_{red} \\ F^{\uparrow} \leftarrow F^{\downarrow}, \;\; \#F^{\downarrow} \leftarrow \#F^{\uparrow} \end{array} \tag{9}$$

The following rules can trigger only once when there is a mistake in set declaration:

$$|F^{\downarrow}| > \#F^{\downarrow} \;\Rightarrow_{red}\; \#F^{\downarrow} \leftarrow |F^{\downarrow}| \tag{10}$$
$$|F^{\uparrow}| < \#F^{\uparrow} \;\Rightarrow_{red}\; \#F^{\uparrow} \leftarrow |F^{\uparrow}| \tag{11}$$

C. Set Constraints

Rule 12 filters 3 set variables F, G, H linked by a difference constraint. Note that multiple assignments in the right-hand side of a rule are made simultaneously. Note also that $min\{a_1, \ldots, a_n\}$ returns the smallest integer a_i.

$$\begin{array}{c} H = F \setminus G \\ \Rightarrow_{red} \\ \left\{ \begin{array}{lcl} H^{\uparrow} & \leftarrow & (H^{\uparrow} \cap F^{\uparrow}) \setminus G^{\downarrow} \\ F^{\uparrow} & \leftarrow & F^{\uparrow} \cap (H^{\uparrow} \cup G^{\uparrow}) \\ G^{\uparrow} & \leftarrow & G^{\uparrow} \setminus H^{\downarrow} \\ H^{\downarrow} & \leftarrow & H^{\downarrow} \cup (F^{\downarrow} \setminus G^{\uparrow}) \\ F^{\downarrow} & \leftarrow & H^{\downarrow} \cup F^{\downarrow} \\ G^{\downarrow} & \leftarrow & G^{\downarrow} \\ \#H^{\downarrow} & \leftarrow & \max\{\#H^{\downarrow}, \; |H^{\downarrow} \cup (F^{\downarrow} \setminus G^{\uparrow})|\} \\ \#F^{\downarrow} & \leftarrow & \max\{\#F^{\downarrow}, \; |H^{\downarrow} \cup F^{\downarrow}|\} \\ \#G^{\downarrow} & \leftarrow & \#G^{\downarrow} \\ \#H^{\uparrow} & \leftarrow & \min\{\#H^{\uparrow}, \; |(F^{\uparrow} \cap H^{\uparrow}) \setminus G^{\downarrow}|\} \\ \#F^{\uparrow} & \leftarrow & \min\{\#F^{\uparrow}, \; |F^{\uparrow} \cap (H^{\uparrow} \cup G^{\uparrow})|\} \\ \#G^{\uparrow} & \leftarrow & \min\{\#G^{\uparrow}, \; |G^{\uparrow} \setminus H^{\downarrow}|\} \end{array} \right. \end{array} \tag{12}$$

Note that after applying Rule 12, if $F^{?} \cap G^{\uparrow} = \emptyset$ the constraint $H = F \setminus G$ is always true and thus, it can be removed. We have such rules for each type of constraint to reduce the CSP instance, and thus the generated SAT instance.

We also add some redundant rules: they do not modify the reduction strenght. They are specialization of some more general rules that can be applied more efficiently. For example, lets consider the constraint $H = F \cap G$. The general rule is:

$$\begin{array}{c} H = F \cap G \\ \Rightarrow_{red} \\ \left\{ \begin{array}{lcl} H^{\uparrow} & \leftarrow & H^{\uparrow} \cap F^{\uparrow} \cap G^{\uparrow} \\ H^{\downarrow} & \leftarrow & H^{\downarrow} \cup (F^{\downarrow} \cap G^{\downarrow}) \\ F^{\downarrow} & \leftarrow & F^{\downarrow} \cup H^{\downarrow} \\ G^{\downarrow} & \leftarrow & G^{\downarrow} \cup H^{\downarrow} \\ F^{\uparrow} & \leftarrow & F^{\uparrow} \setminus (G^{\downarrow} \setminus H^{\uparrow}) \\ G^{\uparrow} & \leftarrow & G^{\uparrow} \setminus (F^{\downarrow} \setminus H^{\uparrow}) \\ \#F^{\downarrow} & \leftarrow & max\{\#F^{\downarrow}, |(F^{\downarrow} \cup H^{\downarrow})|\} \\ \#G^{\downarrow} & \leftarrow & max\{\#G^{\downarrow}, |(G^{\downarrow} \cup H^{\downarrow})|\} \\ \#H^{\downarrow} & \leftarrow & max\{\#H^{\downarrow}, \; |(H^{\downarrow} \cup (F^{\downarrow} \cap G^{\downarrow}))|\} \\ \#F^{\uparrow} & \leftarrow & min\{\#F^{\uparrow}, |(F^{\uparrow} \setminus (G^{\downarrow} \setminus H^{\uparrow}))|\} \\ \#G^{\uparrow} & \leftarrow & min\{\#G^{\uparrow}, |(G^{\uparrow} \setminus (F^{\downarrow} \setminus H^{\uparrow}))|\} \\ \#H^{\uparrow} & \leftarrow & min\{\#H^{\uparrow}, \#F^{\uparrow}, \#G^{\uparrow}, |(H^{\uparrow} \cap F^{\uparrow} \cap G^{\uparrow})|\} \end{array} \right. \end{array} \tag{13}$$

Although equivalent, when $H = \emptyset$ the rule is much simpler, and in practice, it requires less computation and less tests to be applied:

$$\begin{array}{c} \emptyset = F \cap G \\ \Rightarrow_{red} \\ \left\{ \begin{array}{lcl} F^{\uparrow} & \leftarrow & F^{\uparrow} \setminus G^{\downarrow} \\ G^{\uparrow} & \leftarrow & G^{\uparrow} \setminus F^{\downarrow} \\ \#F^{\uparrow} & \leftarrow & \min\{\#F^{\uparrow}, |F^{\uparrow} \setminus G^{\downarrow}|\} \\ \#G^{\uparrow} & \leftarrow & \min\{\#G^{\uparrow}, |G^{\uparrow} \setminus F^{\downarrow}|\} \end{array} \right. \end{array} \quad (14)$$

IV. Encoding Rules

The $\Leftrightarrow_{enc}$ encoding rules aim at transforming CSP set constraints into SAT clauses. Our rules work both on reduced or not reduced CSP set constraints, without generating unuseful clauses.

A. Elements

This encoding rule enforces each element variable to have one and only one value from its domain:

$$\begin{array}{c} Element(v, D_v) \\ \Leftrightarrow_{enc} \\ \forall x \in D_v,\ x_v \rightarrow \bigwedge_{y \in D_v, x \neq y} \neg y_v \quad and \quad \bigvee_{x \in D_v} x_v \\ \Leftrightarrow \\ (\bigwedge_{x \in D_v} \bigwedge_{y \in D_v, y > x} (\neg x_v \vee \neg y_v)) \wedge \bigvee_{x \in D_v} x_v \end{array}$$

This encoding generates $|D_v|.(|D_v|-1)/2$ binary clauses and 1 $|D_v| - ary$ clause.

B. Sets

For a declared set $F :: (F^{\downarrow}, F^{\uparrow}, \#F^{\downarrow}, \#F^{\uparrow})$, the encoding consists in creating the variables and setting to true elements of the set variable lower bound. Consider an element x from the universe, we denote by $?x_F$ the creation of the variable x_F representing the membership of x in the set F.

$$\begin{array}{c} F :: (F^{\downarrow}, F^{\uparrow}, \#F^{\downarrow}, \#F^{\uparrow}) \\ \Leftrightarrow_{enc} \\ \left\{ \begin{array}{l} \forall x \in F^{\uparrow},\ ?x_F \\ \forall x \in F^{\downarrow},\ x_F \end{array} \right. \end{array}$$

C. Set Intersection

In order to be complete, we consider all cases w.r.t. $H^{\uparrow}, H^{\downarrow}, F^{\uparrow}, F^{\downarrow}, G^{\uparrow}, G^{\downarrow}$, even impossible or unuseful cases (noted $-$). Table I lists all these cases.

After reduction, only the 4 cases where binary clauses are generated are not obsolete, the others become unuseful. However, these generic encoding rules permit to handle all models whatever the kind of reduction rules that were applied or not.

V. Experimental Results

All experiments were realized on an Intel® Xeon® Processor E5-2670 with 2.3GHz and 230 GB RAM. Practically, $\Leftrightarrow_{enc}$ rules have been implemented with C^{++} and $\Rightarrow_{red}$ rules as Constraint Handling Rules (CHR [21]). We use Minisat [19] as the SAT solver for all our experiments.

A. Sports Tournament Scheduling Problems

This problem was proposed by Toby Walsh (problem number 26 of the CSPLib [18]) as: "The problem is to schedule a tournament of n teams over $n-1$ weeks, with each week divided into $n/2$ periods, and each period divided into two slots. The first team in each slot plays at home, whilst the second plays the first team away. A tournament must satisfy the following three constraints: every team plays once a week; every team plays at most twice in the same period over the tournament; every team plays every other team.". The n value permits to totally define a Sports Tournament Scheduling instance. A set constraints model with $w = n-1$ and $p = n/2$ can be:

- Universe and set of teams: $\mathcal{U} :: \{1, \ldots, n\} \qquad \mathcal{T} :: \mathcal{U}, \mathcal{U}, n, n$
- Matches of 2 teams for each week and each time period: $\forall i \in [1..w], \forall j \in [1..p],\ \ G_{i,j} :: \perp, \mathcal{U}, 2, 2$
- Every team plays every week: $\forall i \in [1..w],\ \mathcal{T} = \bigcup_{j=1}^{p} G_{i,j}$
- Each team plays at least twice in the same period: $\forall q \in [1..p],\ \forall i \in [1..w-2],\ \forall j \in [i+1..w-1],\ \forall k \in [j+1..w], \emptyset = G_{i,p} \cap G_{j,p} \cap G_{k,p}$
- Every team plays every other team. Since each team already plays each of the $n-1$ week, it is sufficient to enforce that each pair of matches are different (i.e., they can share at most one team): $\forall i \in [1..w-1],\ \forall j \in [i+1..w],\ \forall p_1, p_2 \in [1..p],\ \ G_{i,p_1} \neq G_{j,p_2}$.
- Symmetry breaking 1. The first week is filled simply: team 1 plays team 2 in the first period, team 3 plays team 4 in the second period, ...: $\forall i \in [1..n],\ \ i \in G_{1,((i-1)div2)+1}$
- Symmetry breaking 2. The first team is placed for p weeks (in diagonal, starting from the second week): $\forall i \in [1..p],\ \ 1 \in G_{i+1,i}$

B. Efficiency of SAT pre-processes

To minimize the size of the CNF instances, the use of pre-process is highly recommended. It is also known that minimized instances may not be easier to solve than unrefined instances. Indeed, it is possible that easy to reach solutions are removed by the pre-process and only hard to reach solutions are conserved. We use SatElite [22] as CNF minimizer. We can use it either as complete minimizers (CM_{Sat}) using subsumption, self-subsuming resolution, and variable elimination by substitution... or as an initial unit propagation process (UP_{Sat}). Comparisons are realized on the encoded problems and results are presented in Table II. Unrefined model is the model obtain after $\Leftrightarrow_{enc}$ process. For each instance a reduced instance is associated (obtained by $\Rightarrow_{red}$). For a reduced instance, the suffix "_R" is added to its name.

Table II shows that for unrefined instances, the use of unit propagation or CNF minimizer drastically reduces the number of clauses and variables. Regarding the impact of the reduction rules (reduced instances _R), we can notice that unit propagation cannot produce really smaller instances that

TABLE I
ENCODING RULES FOR SET INTERSECTION.

$$H = F \cap G \quad \Leftrightarrow_{enc}$$

$\forall x \in \mathcal{U}$	$x \in H^{\downarrow}$	$x \in F^{\downarrow}$	$x \in G^{\downarrow}$	$true$
			$x \in G^{?}$	x_G $\quad \lvert H^{\downarrow} \cap F^{\downarrow} \cap G^{?} \rvert$ unit clauses
			$x \notin G^{\uparrow}$	$false$
		$x \in F^{?}$	$x \in G^{\downarrow}$	x_F $\quad \lvert H^{\downarrow} \cap F^{?} \cap G^{\downarrow} \rvert$ unit clauses
			$x \in G^{?}$	$x_F \wedge x_G$ $\quad \lvert H^{\downarrow} \cap (F^{\uparrow} \setminus F^{\downarrow}) \cap G^{?} \rvert$ $\times 2$ unit clauses
			$x \notin G^{\uparrow}$	$false$
		$x \notin F^{\uparrow}$	$x \in G^{\downarrow}$	$false$
			$x \in G^{?}$	$false$
			$x \notin G^{\uparrow}$	$false$
	$x \in H^{?}$	$x \in F^{\downarrow}$	$x \in G^{\downarrow}$	x_H $\quad \lvert H^{?} \cap F^{\downarrow} \cap G^{\downarrow} \rvert$ unit clauses
			$x \in G^{?}$	$x_H \leftrightarrow x_G$ $\quad \lvert H^{?} \cap F^{\downarrow} \cap G^{?} \rvert$ $\times 2$ binary clauses
			$x \notin G^{\uparrow}$	$\neg x_H$ $\quad \lvert H^{?} \cap (F^{\downarrow} \setminus G^{\uparrow}) \rvert$ unit clauses
		$x \in F^{?}$	$x \in G^{\downarrow}$	$x_H \leftrightarrow x_G$ $\quad \lvert H^{?} \cap F^{?} \cap G^{\downarrow} \rvert$ $\times 2$ binary clauses
			$x \in G^{?}$	$x_H \leftrightarrow x_F \wedge x_G$ $\quad \lvert H^{?} \cap F^{?} \cap G^{?} \rvert$ $\times$(2 binary clauses *and* 1 unit clause)
			$x \notin G^{\uparrow}$	$\neg x_H$ $\quad \lvert H^{?} \cap (F^{?} \setminus G^{\uparrow}) \rvert$ unit clauses
		$x \notin F^{\uparrow}$	$x \in G^{\downarrow}$	$\neg x_H$ $\quad \lvert H^{?} \cap G^{\downarrow} \setminus F^{\uparrow} \rvert$ unit clauses
			$x \in G^{?}$	$\neg x_H$ $\quad \lvert H^{?} \cap G^{?} \setminus F^{\uparrow} \rvert$ unit clauses
			$x \notin G^{\uparrow}$	$\neg x_H$ $\quad \lvert H^{?} \setminus F^{\uparrow} \setminus G^{\uparrow} \rvert$ unit clauses
	$x \notin H^{\uparrow}$	$x \in F^{\downarrow}$	$x \in G^{\downarrow}$	$false$
			$x \in G^{?}$	$\neg x_G$ $\quad \lvert F^{\downarrow} \cap (G^{?} \setminus H^{\uparrow}) \rvert$ unit clauses
			$x \notin G^{\uparrow}$	$-$
		$x \in F^{?}$	$x \in G^{\downarrow}$	$\neg x_F$ $\quad \lvert F^{?} \cap G^{\downarrow} \setminus H^{\uparrow} \rvert$ unit clauses
			$x \in G^{?}$	$\neg x_F \vee \neg x_G$ $\quad \lvert F^{?} \cap G^{?} \setminus H^{\uparrow} \rvert$ binary clauses
			$x \notin G^{\uparrow}$	$-$
		$x \notin F^{\uparrow}$	$x \in G^{\downarrow}$	$-$
			$x \in G^{?}$	$-$
			$x \notin G^{\uparrow}$	$-$

TABLE II
EFFICIENCY OF SAT PRE-PROCESSES

			Model characteristics						Encoding time			Resolving time					
	Instances	$\Rightarrow_{red}$	Unrefined		UP_{Sat}		CM_{Sat}		$\Leftrightarrow_{enc}$	UP_{Sat}	CM_{Sat}	Unrefined		UP_{Sat}		CM_{Sat}	
		sec.	#cl	#var	#cl	#var	#cl	#var	sec.	sec.	sec.	sec.	sum	sec.	sum	sec.	sum
STS	8	-	23 812	5 528	13 794	3 867	6 422	1 040	0.04	0.04	0.28	0.01	**0.05**	0.01	0.09	0.01	0.33
	8_R	0.08	16 606	4 549	13 345	3 902	2 649	426	0.03	0.02	0.26	0.00	**0.11**	0.00	0.13	0.00	0.37
	10	-	77 135	17 170	46 917	12 530	29 233	5 194	0.14	0.12	0.91	3.37	3.51	1.18	**1.44**	5.32	6.37
	10_R	0.11	58 235	14 788	45 918	12 650	13 729	2 352	0.11	0.07	0.74	0.01	**0.23**	0.01	0.30	-	0.97
	12	-	198 198	42 612	124 980	32 297	85 581	15 576	0.35	0.40	3.08	143.96	**144.31**	509.24	509.99	266.26	269.70
	12_R	0.22	148 364	35 188	112 467	29 541	40 188	6 624	0.39	0.22	2.22	0.04	**0.65**	0.03	0.86	0.01	2.84
	14	-	436 541	91 154	280 964	70 707	205 706	38 492	0.75	0.76	8.05	-	-	-	-	-	-
	14_R	0.41	342 805	78 104	257 290	65 772	93 320	15 339	0.60	0.42	5.42	0.08	**1.09**	0.06	1.49	0.04	6.47

those based on unrefined instances. We could therefore deduce that reduction rules only eliminate the creation of unit clauses during the encoding. Nevertheless, we can observe that the use of the CNF minimizer provide very small instances (less than the half of the size of the unrefined instances reduced by the CNF minimizer). We can conclude that reduction rules permit to encode a lot of redundant or subsumed clauses.

In terms of encoding time, the use of the CNF minimizer costs a lot and cannot be run for medium and large instances. Unit propagation is not very time consuming but unit propagation is a classical operation done by SAT solvers It is then redundant in terms of running time to do it before and during the solving process.

In order to quickly solve instances, it seems to be more efficient to use only our $\Rightarrow_{red}$ reduction rules.

C. Results for large instances

Table III shows the great impact of our $\Rightarrow_{red}$ reduction rules on our SAT encoding. Indeed, without reduction the STS instances are solved up to size 12 while with application of reduction rules solutions are found until size 66.As said in Section V-B, minimizing process can complicate the resolution because solutions are deleted. Here, we can observe that this is not the case. Our reduction rules makes the search easier: indeed they keep the structure of the problem without removing solutions, but reducing the search space. Size of instances

TABLE III
RESULTS FOR LARGE STS INSTANCES

Inst.	Initial					reduced					
	Unrefined		$\Leftrightarrow_{enc}$	Resolution		$\Rightarrow_{red}$	Unrefined		$\Leftrightarrow_{enc}$	Resolution	
	#cl $\times 10^3$	#var$\times 10^3$	sec.	sec.	sum	sec.	#cl$\times 10^3$	#var$\times 10^3$	sec.	sec.	sum
8	24	6	0.02	0.04	**0.06**	0.08	15	4	0.02	0.02	0.12
10	77	17	0.01	9.30	9.31	0.11	54	13	0.01	0.04	**0.16**
12	198	43	0.03	387.85	387.88	0.22	148	35	0.02	0.14	**0.38**
14	437	91	0.04	-	-	0.41	343	78	0.03	0.22	**0.66**
16	861	175	0.09	-	-	0.66	700	154	0.05	0.45	**1.16**
18	1 569	314	0.17	-	-	0.98	1 307	281	0.14	0.89	**2.01**
20	2 676	528	0.32	-	-	1.34	2 275	479	0.24	1.69	**3.28**
22	4 331	842	0.49	-	-	1.92	3 742	773	0.41	3.43	**5.76**
24	6 713	1 289	0.75	-	-	2.65	5 879	1 194	0.66	4.82	**8.13**
26	10 041	1 905	1.15	-	-	3.71	8 890	1 779	0.98	9.24	**13.93**
28	14 567	2 735	1.62	-	-	5.34	13 020	2 569	1.37	12.37	**19.08**
30	20 591	3 827	2.26	-	-	6.13	18 551	3 616	2.03	20.20	**28.36**
32	28 453	5 241	3.18	-	-	8.11	25 813	4 974	2.92	31.93	**42.96**
34	38 581	7 059	4.00	-	-	10.62	35 203	6 719	3.89	36.17	**50.68**
36	51 396	9 343	5.54	-	-	12.67	47 151	8 925	5.04	83.38	**101.09**
38	67 397	12 178	7.26	-	-	14.96	62 129	11 668	6.64	124.25	**145.85**
40	87 144	15 655	9.29	-	-	12.78	80 678	15 041	8.44	114.74	**135.96**
50	266 070	46 637	27.93	-	-	36.93	250 325	45 254	27.38	880.50	**944.81**
52	323 676	56 497	33.62	-	-	46.99	305 266	54 901	31.79	1240.79	**1319.57**
54	390 835	67 948	40.61	-	-	69.64	369 437	66 116	39.13	1492.06	**1600.83**
56	468 687	81 175	48.36	-	-	60.58	443 952	79 083	46.24	1803.75	**1910.57**
58	558 459	96 375	67.49	-	-	60.19	530 012	93 996	55.61	2362.21	**2478.01**
60	661 467	113 760	70.66	-	-	78.28	628 906	111 067	64.76	2952.74	**3095.78**
62	779 123	133 556	79.58	-	-	87.63	742 017	130 519	77.08	3367.36	**3532.07**
64	912 933	156 005	95.06	-	-	100.48	870 823	152 592	92.14	4903.03	**5095.65**
66	1 064 779	181 500	109.51	-	-	115.36	1 017 067	177 625	105.45	5247.73	**5468.54**
68	1 236 149	210 203	126.70	-	-	135.61	1 182 405	205 875	122.12	-	-
70	1 428 864	242 406	145.71	-	-	154.44	1 368 535	237 589	139.81	-	-
72	1 644 854	278 418	172.92	-	-	182.35	1 577 355	273 071	167.62	-	-

is thus not the only criterion: structure and search size also matter. For example, some reduced larger instances can be larger than some not reduced smaller instances; however, the reduced instances maybe solved while the not reduced one may not. For example, not reduce instance 14 of STS is smaller than reduced instance 20 of STS; however, reduced instance 20 is solved while not reduced instance 14 is not. Note that for instances 68 to 72, Minisat solver stop immediately the search due to the too large number of clauses and variables. Specific solvers [20] were proposed to solved this problem over constraining the problem (by adding extra constraints that are not present in the initial problem). This way, the larger solved instance is 70 but some instances may not have anymore a solution (due to over constraints).

VI. COMPARISONS WITH PREVIOUS WORKS

Compared to [13], the benefits are:

- the modeling language is richer and proposes finite domain variables. Set cardinality, set min and set max constraints now link a finite domain variable to a set variable. The language is more expressive and more practical;
- the $\Rightarrow_{red}$ reduction rules now use upper and lower bounds of sets as well as min and max cardinality. The filtering is much stronger and thus, search spaces are more reduced;
- the $\Leftrightarrow_{enc}$ encoding rules are more generic and can be applied to reduced or not reduced CSP set constraints. They thus generate much smaller SAT instances.

To summarize, we are now able to model more problems, to solve them more efficiently, and to tackle larger instances.

We can compare our work with works about SAT encoding techniques such as [5] and [6]. These works make a relation between CSP solving and SAT solving in terms of properties such as consistencies for finite domain variables and constraints. In this article, we are concerned with a different type of constraints (i.e., set constraints) and we try to obtain small SAT instances that are also well-suited for standard SAT solvers. Moreover, [5] and [6] do not consider a reduction phase as our $\Rightarrow_{red}$ rules.

Our approach is similar to [8] in which alldifferent global constraints and overlapping alldifferent constraints are handled expressively before being encoded automatically into SAT using rewrite rules. Note also that we use the work of [9] about the *cardinality* global constraint in order to perform the encoding of set cardinality.

Since we exploit cardinality as defined in [10], our filtering phase is stronger than the constraint propagation phase of [11]. Stronger filtering can be designed (e.g., [17]) using richer set representations and the length-lex order. Although theoretical results about such constraint propagation algorithms are negative, in practice they may behave better than bound consistency

with cardinality for some benchmarks (such as the Social golfer problem as shown in [17]). However, for our purpose, we prefered a good balance between filtering and genericity. Moreover, the length-lex order is efficient for enumeration, but in our case we do not need enumeration but only filtering.

Some works, such as [23] "compile" set constraints into a Reduced Ordered Binary Decision Diagrams which is directly used for solving the problem. This technique seems efficient and it is claimed that it can be extended to integers and multi-sets. However, we want to stay as close as possible of constraint structures to be able to use various tools and constraint structures to treat these constraints. Moreover, we are also interested in integrating some other global constraints. Finally, our aim is not to solve the model nor to design a solver, but to prepare models in order to obtain a better encoding that will be solved by a SAT solver.

In terms of efficiency, we have shown for the STS problem that our technique is competitive with the best (to our knowledge) ad-hoc-solver [20], i.e., especially designed for the STS problem. Moreover, this approach over constraints the problem, and thus, may lose solutions.

We made some similar tests with the Social Golfer Problem. Using our model, Conjunto [11] can only solve small instances. With the same model, Minizinc is stucked very quickly. We tried some other models with Minizinc (not based on sets) but the results were not better. Thus, we were competitive with generic CSP solvers for the SGP.

VII. Conclusion

We have presented a technique for encoding set constraints into SAT: the modeling process is achieved using some very expressive set constraints; they are then reduced by our $\Rightarrow_{red}$ rules before being automatically converted ($\Leftrightarrow_{enc}$) into SAT variables and clauses. We have illustrated our approach on the Sports Tournament Scheduling problem and we have shown some good results with the application of reduction and encoding rules. We nearly reach the results of the best (to our knowledge) ad-hoc solver [20] which over constraints the problem, and thus, may sometimes not be able to find a solution.

The advantages of our technique are the following:

- the modeling process is simple, expressive, and readable. Moreover, it is solver independent and independent from CSP or SAT solvers;
- the technique is less error-prone than direct SAT encodings;
- the SAT instances which are automatically generated are smaller in terms of number of variables and clauses;
- finally, with respect to solving time, adding reduction process permits to reduce the cumulative running time (reduction+encoding+resolution);
- the generated SAT instances also appeared to be well-suited for Minisat.

In the future, we plan to extend our constraints encoding rules for formalizing finite domain variables arithmetic constraints. To this end, we will need to add some new constraints and to complete our $\Leftrightarrow_{enc}$ and $\Rightarrow_{red}$ rules. Up to now we have our proper model format (XML-like) but we plan to use the XCSP3 [24] standard. We also plan to extend our technique to multisets [25], [26] and sequences.

References

[1] F. Rossi, T. P. van Beek, and Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman & Company, 1979.

[3] M. Triska and N. Musliu, "An improved sat formulation for the social golfer problem," *Annals of Operations Research*, vol. 194, no. 1, pp. 427–438, 2012.

[4] I. Gent and I. Lynce, "A sat encoding for the social golfer problem," in *IJCAI'05 workshop on modelling and solving problems with constraints*, 2005.

[5] F. Bacchus, "Gac via unit propagation," in *Proc. of CP 2007*, ser. LNCS, vol. 4741. Springer, 2007, pp. 133–147.

[6] C. Bessière, E. Hebrard, and T. Walsh, "Local consistencies in sat," in *Selected Revised Papers of SAT 2003.*, ser. LNCS, vol. 2919. Springer, 2004, pp. 299–314.

[7] J. Petke and P. Jeavons, *The Order Encoding: From Tractable CSP to Tractable SAT*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 371–372.

[8] F. Lardeux, E. Monfroy, F. Saubion, B. Crawford, and C. Castro, "Sat encoding and csp reduction for interconnected alldiff constraints," in *Proc. of MICAI 2009*, 2009, pp. 360–371.

[9] O. Bailleux and Y. Boufkhad, "Efficient cnf encoding of boolean cardinality constraints," in *Proc. of CP 2003*, vol. 2833. Springer, 2003, pp. 108–122.

[10] F. de Moura e Castro Ascensão de Azevedo, "Constraint solving over multi-valued logics - application to digital circuits," Ph.D. dissertation, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2002.

[11] C. Gervet, "Conjunto: Constraint propagation over set constraints with finite set domain variables," in *Proc. of ICLP'94*. MIT Press, 1994, p. 733.

[12] "CHOCO," http://www.emn.fr/z-info/choco-solver/.

[13] F. Lardeux, E. Monfroy, B. Crawford, and R. Soto, "Set constraint model and automated encoding into SAT: application to the social golfer problem," *Annals OR*, vol. 235, no. 1, pp. 423–452, 2015.

[14] F. Lardeux and E. Monfroy, "From declarative set constraint models to "good" SAT instances," in *Artificial Intelligence and Symbolic Computation - 12th International Conference, AISC 2014, Seville, Spain, Dec. 11-13*, 2014, pp. 76–87.

[15] K. Apt, *Principles of Constraint Programming*. Cambridge University Press, 2003.

[16] F. Azevedo, "Cardinal: A finite sets constraint solver," *Constraints*, vol. 12, no. 1, pp. 93–129, 2007.

[17] J. Yip and P. V. Hentenryck, "Checking and filtering global set constraints," in *Principles and Practice of Constraint Programming - CP 2011, Perugia, Italy, September 12-16*, 2011, pp. 819–833.

[18] T. Walsh, "CSPLib problem 026: Sports tournament scheduling," http://www.csplib.org/Problems/prob026.

[19] N. Eén and N. Sörensson, "An extensible sat-solver," in *SAT 2003*, vol. 2919, 2003, pp. 502–518.

[20] J. Hamiez and J. Hao, "A note on a sports league scheduling problem," *CoRR*, vol. abs/1410.2721, 2014. [Online]. Available: http://arxiv.org/abs/1410.2721

[21] T. Früwirth, *Constraint Handling Rules*. Cambridge University Press, 2009.

[22] N. Eén and A. Biere, "Effective preprocessing in sat through variable and clause elimination," in *SAT 2005*, vol. 3569, 2005, pp. 61–75.

[23] P. Hawkins, V. Lagoon, and P. J. Stuckey, "Solving set constraint satisfaction problems using robdds," *J. Artif. Intell. Res. (JAIR)*, vol. 24, pp. 109–156, 2005.

[24] F. Boussemart, C. Lecoutre, C. Piette, and V. Perradin, "XCSP3 an integrated format for benchmarking combinatorial constrained problems," http://www.xcsp.org/.

[25] T. Walsh, *Consistency and Propagation with Multiset Constraints: A Formal Viewpoint*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 724–738.
[26] Y. C. Law, J. H. M. Lee, T. Walsh, and M. H. C. Woo, "Multiset variable representations and constraint propagation," *Constraints*, vol. 18, no. 3, pp. 307–343, 2013.