From Declarative Set Constraint Models to "Good" SAT Instances

Frédéric Lardeux, Eric Monfroy

From Declarative Set Constraint Models to "Good" SAT Instances

Frédéric Lardeux[1] and Eric Monfroy[2]

[1] Université d'Angers, France. Frederic.Lardeux@univ-angers.fr
[2] LINA, UMR CNRS 6241, TASC INRIA, Université de Nantes, France
Eric.Monfroy@univ-nantes.fr

Abstract. On the one hand, Constraint Satisfaction Problems allow one to declaratively model problems. On the other hand, propositional satisfiability problem (SAT) solvers can handle huge SAT instances. We thus present a technique to declaratively model set constraint problems, to reduce them, and to encode them into "good" SAT instances. We illustrate our technique on the well-known nqueens problem. Our technique is simpler, more expressive, and less error-prone than direct hand modeling. The SAT instances that we automatically generate are rather small w.r.t. hand-written instances.

1 Introduction

Most of combinatorial problems can be formulated as Constraint Satisfaction Problems (CSP) [**?**]. 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. One of the main strength of CSP is declarativity and expressiveness: 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), ...

On the other hand, the propositional satisfiability problem (SAT) [**?**] is restricted (in terms of expressiveness) to Boolean variables and propositional formulae. Coding set constraints directly into SAT is a tedious tasks (see for example [**?**] or [**?**]). 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., [**?**,**?**]) in order to benefit from the declarativity and expressiveness of CSP and the power of SAT, and 2) introduce more declarativity into SAT, e.g., with global constraints such as alldifferent [**?**], or cardinality [**?**].

Various systems of set constraints (either specialized systems [**?**], libraries for constraint programming systems such as [**?**], or the set constraint library of

CHOCO [?]) 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 the transformation of set constraints into SAT instances: we often refer to this transformation as "encoding". In [?], we presented encoding rules that are directly applied on the CSP set constraints. However, we have noticed that some supports of sets (i.e., elements that are possibly in this set) could be reduced (some elements can be removed from the supports without loosing any solution). Thus, the generated SAT instances include non necessary information.

It is inconceivable to force the user to write reduced CSP models: first, because it is a tedious and error-prone task; and second, it may be impossible to see all the relations between the sets (more especially when working on sets which supports are not yet declared). Thus, our approach consists in providing:

- a simple but complete, declarative, and expressive set language for easily modeling problems with constraints such as intersection, union, cardinal of sets, ...
- a set of reduction rules ($\Rightarrow_{red}$) to reduce CSP models. In fact, these rules define constraint propagation [?] for sets and elements and make the model Generalized Arc Consistent [?].
- a set of encoding rules ($\Leftrightarrow_{enc}$) that convert CSP constraints into SAT instances.

In this paper, we illustrate our approach with the famous nqueen problem. Moreover, we have tried our technique on various problems (e.g., Social Golfer Problem [?], Sudoku, Car-sequencing) and the SAT instances which are automatically generated have a complexity similar to the complexity of improved hand-written SAT formulations, and their solving with a SAT solver (in our case Minisat [?]) is efficient compared to other SAT approaches.

Compared to [?], the $\Rightarrow_{red}$ reduction rules enable us:

- to simplify the encoding rules ($\Leftrightarrow_{enc}$): indeed, some transformation cases become unuseful; the encoding rules become even more simple and readable;
- to obtain even smaller SAT instances, in terms of clauses and variables; these problems are solved faster;
- to tackle and solve larger problems that we were unable to encode using only our previous encoding rules (for size reasons).

We can compare our work with SAT encoding techniques such as [?] and [?]. 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. Our approach is similar to [?] in which alldifferent global constraints and overlapping alldifferent constraints are handled declaratively before being encoded automatically in SAT using rewrite rules. Note also that we

use the work of [?] about the *cardinality* global constraint in order to perform the encoding of set cardinality. Our goal is not to compete with standard set solvers, but to introduce set constraints into SAT.

In the next section (Section **??**) we present the CSP set constraint language. In Section **??** we show the rules to reduce models. Section **??** presents our new rule-based system for encoding set constraints into SAT. Section **??** illustrates our approach on the nqueen problem. We finally conclude in Section **??**.

2 Set Constraint

2.1 Universe and Supports

In order to encode set constraints into SAT, we consider 3 notions: *universe*, *support*, and *domain*. Informally, the universe is the set of all elements that are considered in a model of a given problem; the support $\mathcal{F}$ of a set F appearing in this model is a set of possible elements of F (i.e., $\mathcal{F}$ is a superset of F); and the domain of an element variable is a set of possible values for this element.

Definition 1. *Let P be a problem, and M be a model of P in $\mathcal{L}$, i.e., a description of P from the natural language to the language of constraints $\mathcal{L}$.*

- *The universe $\mathcal{U}$ of M is a finite set of constants.*
- *The support $\mathcal{F}$ of the set F of the model M is a subset of the universe $\mathcal{U}$. $\mathcal{F}$ represents the constants of $\mathcal{U}$ that can possibly be elements of F:*

$$F \subseteq \mathcal{F} \subseteq \mathcal{U} \quad and \quad F \in \mathcal{P}(\mathcal{F})$$

  where $\mathcal{P}(\mathcal{F}) = \{A | A \subseteq \mathcal{F}\}$ is the power set of $\mathcal{F}$. We say that F is over $\mathcal{F}$.
- *The domain D_x of a variable element x is a subset of the universe $\mathcal{U}$; D_x represents the elements of $\mathcal{U}$ that are possible values (i.e., constants) for x.*

Note that each element of $\mathcal{U} \setminus \mathcal{F}$ cannot be an element of F. In the following, we denote sets by upper-case letters (e.g., F) and their supports by calligraphic upper-case letters (e.g., $\mathcal{F}$). Variable elements are represented by lower-case letters (e;g., x) and their domain by D indexed by the variable name (e.g., D_x). When there is no confusion, we shorten "the set F of the model M" to "F".

Consider a model M with a universe $\mathcal{U}$, and a set F over $\mathcal{F}$. For each element x of $\mathcal{F}$, we consider a Boolean variable $x_\mathcal{F}$ which is true if $x \in F$ and false otherwise. We call the set of such variables the support variables for F in $\mathcal{F}$. In the following, we write $x_\mathcal{F}$ for $x_\mathcal{F} = true$ and $\neg x_\mathcal{F}$ for $x_\mathcal{F} = false$.

Example 1. Let $\mathcal{U} = \{x, y, z, t\}$ be the universe of a model M, and $\mathcal{F} = \{x, y, t\}$ be the support of a set F of M. Then, we have 3 Boolean variables $x_\mathcal{F}$, $y_\mathcal{F}$, and $t_\mathcal{F}$ corresponding respectively to x, y, and t to represent F. By definition, $z \notin F$ and there is no $z_\mathcal{F}$ variable; and x, y, t can possibly be in F. Consider now that $F = \{x, y\}$. Then, we have $x_\mathcal{F}$, $y_\mathcal{F}$, and $\neg t_\mathcal{F}$.

2.2 Syntax

In order to declare objects, we use the following declarations:

- $Universe(U)$ is used to declare the universe as the set U;
- $Set(F, \mathcal{F})$: declares a set F together with its support $\mathcal{F}$.;
- $Element(x, D_x)$: creates a variable x of type element with its domain D_x.

Consider F, G, H, and F_i (i ranging from 1 to n) being sets, and x being an element. We consider the following usual (CSP) set constraints:

element (dis)equality	$x = y$	$(x \neq y)$
(non)membership	$x \in F$	$(x \notin F)$
set (dis)equality	$F = G$	$(F \neq G)$
intersection	$H = F \cap G$	
union	$H = F \cup G$	
inclusion	$F \subseteq G$	
difference	$H = F \setminus G$	
multi-intersection	$F = \bigcap_{i=1}^{n} F_i$	
multi-union	$F = \bigcup_{i=1}^{n} F_i$	
cardinality$\{=, <, >\}$	$\lvert F \rvert \{=, <, >\} k$	

More constraints could be defined, but they can be deduced from these basic constraints. A model for a problem is given by:

1. a universe;
2. some sets together with their supports;
3. some variable elements with their domains;
4. some constraints between sets and elements.

3 Reducing supports

Support sizes are a crucial parameter for the sizes of generated SAT instances. Moreover, it is quite complicated (and sometimes impossible) to write a model with "reduced" supports. For example, consider 3 sets: $Set(G, \{1, 2, \ldots, 10000\})$, $Set(F, \{9999, \ldots, 20000\})$, and $Set(H, \{5000, \ldots, 25000\})$. Latter in the model, let consider that the constraint $H = F \cup G$ appear. Then, the support of H can be reduced to $\{5000, \ldots, 20000\}$, and the support of G can be reduced to $\{5000, \ldots, 10000\}$.

We thus consider some reduction rules $\Rightarrow_{red}$ to reduce domains and supports w.r.t. constraints. These rules remove elements of the supports and domains that cannot participate in any solution to the problem. We first start with failure case, i.e., cases that do not lead to any solution.

Failures: Rule **??** causes a fail when the domain of a variable is empty. Rule **??** leads to a fail when the imposed cardinality is higher than the size of the support

of the set. Rule **??** is similar for inequality about cardinal.

$$D_x = \emptyset \;\Rightarrow_{red}\; fail \tag{1}$$
$$|F| = k \;\Rightarrow_{red}\; fail \;\text{ if }\; |\mathcal{F}| < k \tag{2}$$
$$|F| > k \;\Rightarrow_{red}\; fail \;\text{ if }\; |\mathcal{F}| \leq k \tag{3}$$

Domain reduction: Rule **??** reduces the domains of two equal variables. When 2 variables are disequal, Rule **??** reduces the domain of the second variable when the domain of the first one is restricted to a singleton ($\{v_x\}$). The domain of a variable x is reduced by Rule **??** w.r.t. the support of a set F in which x must appear (constraint $x \in F$):

$$x = y \;\Rightarrow_{red}\; \begin{cases} D_x \leftarrow D_x \cap D_y, \\ D_y \leftarrow D_x \cap D_y \end{cases} \tag{4}$$
$$x \neq y, D_x = \{v_x\} \;\Rightarrow_{red}\; D_y \leftarrow D_y \setminus \{v_x\} \tag{5}$$
$$x \in F \;\Rightarrow_{red}\; D_x \leftarrow D_x \cap \mathcal{F} \tag{6}$$

Support reduction: When 2 sets must be equal, Rule **??** reduces their supports to their intersection. Intersection constraint enables to reduce the domain of the set intersection (Rule **??**) whereas union constraint may reduce the supports of the 3 sets appearing in the constraint (Rule **??**). Inclusion constraint only reduces the support of the included set (Rule **??**). Difference constraint may reduce 2 supports of the 3 sets (Rule **??**). Rules **??** and **??** are similar to Rules **??** and **??** for multi-union and multi-intersection constraints.

$$F = G \;\Rightarrow_{red}\; \mathcal{F} \leftarrow \mathcal{F} \cap \mathcal{G},\;\; \mathcal{G} \leftarrow \mathcal{G} \cap \mathcal{F} \tag{7}$$
$$H = F \cap G \;\Rightarrow_{red}\; \mathcal{H} \leftarrow \mathcal{H} \cap \mathcal{F} \cap \mathcal{G} \tag{8}$$
$$H = F \cup G \;\Rightarrow_{red}\; \mathcal{H} \leftarrow \mathcal{H} \cap (\mathcal{F} \cup \mathcal{G}),\;\; \mathcal{F} \leftarrow \mathcal{F} \cap \mathcal{H},\;\; \mathcal{G} \leftarrow \mathcal{G} \cap \mathcal{H} \tag{9}$$
$$F \subseteq G \;\Rightarrow_{red}\; \mathcal{F} \leftarrow \mathcal{F} \cap \mathcal{G} \tag{10}$$
$$H = F \setminus G \;\Rightarrow_{red}\; \mathcal{H} \leftarrow \mathcal{H} \cap \mathcal{F},\;\; \mathcal{F} \leftarrow \mathcal{F} \cap \mathcal{H} \tag{11}$$
$$H = \bigcup_{i=1}^{n} F_i \;\Rightarrow_{red}\; \mathcal{H} \leftarrow \mathcal{H} \cap (\bigcup_{i=1}^{n} \mathcal{F}_i),\;\; \forall i \in [1..n]\; \mathcal{F}_i \leftarrow \mathcal{F}_i \cap \mathcal{H}, \tag{12}$$
$$H = \bigcap_{i=1}^{n} F_i \;\Rightarrow_{red}\; \mathcal{H} \leftarrow \mathcal{H} \cap (\bigcap_{i=1}^{n} \mathcal{F}_i) \tag{13}$$

Rule application: $\Rightarrow_{red}$ rules can be seen as filtering (or reduction) functions in constraint programming. They can thus be applied by a fixed-point algorithm such as chaotic iterations [**?**,**?**,**?**]: since the rules have the required properties (monotonic decreasing and idempotent), termination is ensured.

In fact, these rules define constraint propagation for sets and elements. Moreover, they enforce GAC (Generalised Arc Consistency [**?**]), i.e., the supports and domains cannot be reduced anymore using a single constraint without loosing

solution local to this constraint. Since this is not the focus of this paper, we don't give here the proof, but just the basis: with respect to GAC, variable domains (in terms of constraint programming) are the domains of the variable elements, and the power-set of the supports for sets.

4 The $\Leftrightarrow_{enc}$ Encoding Rules

We can now define the encoding of our CSP set constraints into SAT. In the following, we consider three sets F, G, and H respectively defined on the supports $\mathcal{F}$, $\mathcal{G}$ and $\mathcal{H}$ of the universe $\mathcal{U}$, and for each $x \in \mathcal{U}$ the various Boolean variables $x_{\mathcal{F}}$, $x_{\mathcal{G}}$, and $x_{\mathcal{H}}$ as defined before. $|G|$ denotes the cardinality of the set G.

Contrary to [**?**], we consider here that supports and domains are reduced using $\Rightarrow_{red}$ rules. Allowing the supports to be non reduced eases the modeling process: indeed, one does not have to compute the reduced support and can use a superset of it or the universe; then, supports are reduced automatically by the $\Rightarrow_{red}$ rules and the $\Leftrightarrow_{enc}$ encoding rules can generate smaller SAT instances.

The clauses that are generated by these rules are of the form $\forall x \in \mathcal{F},\ \phi(x_{\mathcal{F}})$ which denotes the $|\mathcal{F}|$ formulae $\phi(x_{\mathcal{F}})$ built for each element x of the support $\mathcal{F}$ of F (x refers to the element of the universe/support, and $x_{\mathcal{F}}$ to the variable representing x for the set F).

$Element(x, D_x)$ and $set(F, \mathcal{F})$ enable to create the required SAT variables: as many variables as the support for a set, and as many as the domain for a variable element. In the following, we present rules for set constraint encodings with: first, the set constraint, then its encoding in SAT (i.e., some clauses linking the SAT variables), and finally, the number of clauses generated.

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

$$Element(v, D_v) \Leftrightarrow_{enc} \forall x \in D_v,\ \bigvee_{x \in D_v} (\wedge_{y \in D_v, x \neq y} (\neg y_v) \wedge x_v) \quad |D_v|^2 \text{ bin. clauses}$$

Element variable (dis)equality let us recall that after application of $\Rightarrow_{red}$ rules on $v = w$, v and w have the same domain. This is not the case for $v \neq w$.

$$v = w \quad \Leftrightarrow_{enc} \quad \forall x \in D_v,\ x_v \leftrightarrow x_w \qquad 2.|D_v| \text{ binary clauses}$$

$$v \neq w \quad \Leftrightarrow_{enc} \quad \begin{cases} \forall x \in D_v,\ x_v \rightarrow \neg x_w & 2.|D_v| \text{ binary clauses} \\ \forall x \in D_w,\ x_w \rightarrow \neg x_v & 2.|D_w| \text{ binary clauses} \end{cases}$$

Membership Constraint This constraint enforces the element v to be in the set F: if $x \in \mathcal{F}$ (x is in the support of F), then the corresponding support variable must be true (i.e., $x_{\mathcal{F}}$). The constraint $x \notin F$ can be similarly defined.

$$v \in F \ \Leftrightarrow_{enc} \ \forall x \in D_v,\ x_v \rightarrow x_{\mathcal{F}} \qquad |D_v| \text{ binary clauses}$$

$$v \notin F \ \Leftrightarrow_{enc} \ \forall x \in D_v \cap \mathcal{F},\ x_v \rightarrow \neg x_{\mathcal{F}} \ \wedge \ x_{\mathcal{F}} \rightarrow \neg x_v \qquad 2.|D_v| \text{ binary clauses}$$

Set (Dis)Equality Constraint After reduction, 2 equal sets G and F have the same support. Thus, the encoding for the equality constraint is:

$$F = G \quad \Leftrightarrow_{enc} \quad \forall x \in \mathcal{F},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{G}} \qquad 2.|\mathcal{F}| \text{ binary clauses}$$

The constraint $F \neq G$ is satisfied when at least one variable of the intersection of the 2 sets is different in F and G, or when a variable appearing in the support of F and not in the one of G is true (and vice-versa):

$$F \neq G \Leftrightarrow_{enc} (\bigvee_{x \in \mathcal{F} \cap \mathcal{G}} x_{\mathcal{F}} \leftrightarrow \neg x_{\mathcal{G}}) \vee (\bigvee_{x \in \mathcal{F} \setminus \mathcal{G}} x_{\mathcal{F}}) \vee (\bigvee_{x \in \mathcal{G} \setminus \mathcal{F}} x_{\mathcal{G}})$$
$$2.|F \cup G| \text{ clauses of size } 2 + |\mathcal{F} \cap \mathcal{G}| - |F \cup G|$$

Intersection Constraint Let H be the intersection of two sets G and F: the reduced support of H is included in the intersection of the supports of G and F.

- for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: a support variable of H is true if and only if this variable is in F and G;
- for the elements of $(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}$: since such an element cannot be in H, it must not be in F or in G.

$$F \cap G = H \quad \Leftrightarrow_{enc}$$
$$\begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & |\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ & +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses} \\ \forall x \in (\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H},\ \neg x_{\mathcal{F}} \vee \neg x_{\mathcal{G}} & |(\mathcal{F} \cap \mathcal{G}) \setminus \mathcal{H}| \text{ binary clauses} \end{cases}$$

Union Constraint More cases must be considered for this constraints:

- for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: a support variable of H is true if and only if this variable is in F or in G; this is the trivial case;
- for the elements of $(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}$: this case is a reduction of the previous one but it is however equivalent; since such an element x is not in the support of G then $x_{\mathcal{G}}$ does not exist, and x is in H if and only if it is in F; note that the generated clauses are exactly the same removing $x_{\mathcal{G}}$;
- for the elements of $(\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F}$: this is the symmetrical case for G;

$$F \cup G = H \quad \Leftrightarrow_{enc}$$
$$\begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \vee x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & |\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ & +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses} \\ \forall x \in (\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}} & 2.|(\mathcal{F} \cap \mathcal{H}) \setminus \mathcal{G}| \text{ binary clauses} \\ \forall x \in (\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F},\ x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} & 2.|(\mathcal{G} \cap \mathcal{H}) \setminus \mathcal{F}| \text{ binary clauses} \end{cases}$$

Inclusion Constraint Elements of $\mathcal{F}$ that are in F must also be in G:

$$F \subseteq G \Leftrightarrow_{enc} \forall x \in \mathcal{F},\ x_{\mathcal{F}} \rightarrow x_{\mathcal{G}} \qquad |\mathcal{F}| \text{ binary clauses}$$

Difference Constraint After reduction, F and H have the same support:

- for the elements of $\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}$: such elements are in H if and only if they are in F and not in G;
- for the elements of $\mathcal{F} \setminus \mathcal{G}$: they are in H if and only if they are in F.

$$\begin{array}{ll} H = F \setminus G & \Leftrightarrow_{enc} \\ \begin{cases} \forall x \in \mathcal{F} \cap \mathcal{G} \cap \mathcal{H},\ x_{\mathcal{F}} \wedge \neg x_{\mathcal{G}} \leftrightarrow x_{\mathcal{H}} \\ \forall x \in \mathcal{F} \setminus \mathcal{G},\ x_{\mathcal{F}} \leftrightarrow x_{\mathcal{H}} \end{cases} & \begin{array}{l} |\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ ternary clauses} \\ +2.|\mathcal{F} \cap \mathcal{G} \cap \mathcal{H}| \text{ binary clauses} \\ 2.|\mathcal{F} \setminus \mathcal{G}| \text{ binary clauses} \end{array} \end{array}$$

Multi-union Constraint The multi-union constraint $H = \bigcup_{i=1}^{n} F_i$ is equivalent to the $n-1$ ternary constraints: $F_{1,2} = F_1 \cap F_2$, $F_{1,2,3} = F_{1,2} \cap F_3$, ... It is not only a short-hand, but it also significantly reduces the number of variables (only variables for H are required, not for each set $F_{1,2,\ldots}$) and generated clauses. Indeed, elements of $\bigcap_{i=1}^{n} \mathcal{F}_i$ are considered once in the multi-union constraint whereas they are considered $n-1$ times in the corresponding $n-1$ binary union constraints. the set $\{1, \ldots, n\}$.

$$H = \bigcup_{i=1}^{n} F_i \Leftrightarrow_{enc} \begin{cases} \forall i \in N,\ \forall x \in F_i,\ x_{\mathcal{F}_i} \rightarrow x_{\mathcal{H}} & \sum_{i=1}^{n} |\mathcal{F}i| \text{ binary clauses} \\ \forall x \in \mathcal{H},\ x_{\mathcal{H}} \rightarrow \bigvee_{i \in N, x \in F_i} x_{\mathcal{F}i} & |\mathcal{H}| \ m\text{-ary clauses } (m \leq n) \end{cases}$$

Multi-intersection Constraint Similarly, we define the multi-intersection constraints. As for the multi-union, the advantage is the gain of clauses and variables in the generated SAT instance:

$$\begin{array}{ll} H = \bigcap_{i \in N} F_i & \Leftrightarrow_{enc} \\ \begin{cases} \forall x \in \mathcal{H},\ \bigwedge_{i=1}^{n} x_{\mathcal{F}_i} \leftrightarrow x_{\mathcal{H}} \\ \forall x \in (\bigcap_{i=1}^{n} \mathcal{F}_i) \setminus \mathcal{H},\ \bigvee_{i \in N} (\neg x_{\mathcal{F}_i}) \end{cases} & \begin{array}{l} 2.|\mathcal{H}| \ (n+1)\text{-ary clauses} \\ |\bigcap_{i \in N} \mathcal{F}_i \setminus \mathcal{H}| \ n\text{-ary clauses} \end{array} \end{array}$$

Cardinality Constraint This constraint has been studied for the encoding of global constraints (see e.g., [**?**]). The very intuitive encoding is quite simple but the generated clauses are too large. A more efficient encoding is based on the unary representation of integers (an integer $k \in [0..n]$ is represented by 1 k times followed by 0 $n-k$ times). We re-use this encoding [**?**] that we have chosen for the unit clauses it generates, and thus, the simplifications that can be achieved in the SAT instances. Consider the set G over the support $\mathcal{G}$ of size n, then the set constraint $|G| = k$ generates: $n + \sum_{i=1}^{n} 2u_i^n(\lfloor \frac{u_i^n}{2} \rfloor + 1)(\lceil \frac{u_i^n}{2} \rceil + 1) - (\frac{u_i^n}{2} + 1)$ clauses and $\sum_{i=1}^{n} u_i^n$ variables. with $u_n^n = 1$,$u_1^n = n$ and $u_i^n = u_{2i-1}^n + 2u_{2i}^n + u_{2i+1}^n$. The *cardinality_le* constraint can similarly be generated.

5 Application to the nqueen problem

Practically, the $\Rightarrow_{red}$ rules have been implemented as Constraint Handling Rules (CHR [**?**]), and the $\Leftrightarrow_{enc}$ rules with C^{++}. To illustrate our approach, we have

chosen the nqueen problem for various reasons: it is not well suited for SAT solvers; it scales well; it can be modeled in various ways with sets. We first give an intuitive model and then a more efficient model of the nqueen problem.

nqB: Model with the Board as universe. The variables are the following:

- Universe: $\mathcal{U} = \{x_{1,1}, \ldots, x_{n,n}\}$, i.e., the set of cells of a $n \times n$ board
- Rows: $\forall i \in [1..n], set(R_i, \{x_{i,1}, \ldots, x_{i,n}\})$
- Columns: $\forall i \in [1..n], set(C_i, \{x_{1,i}, \ldots, x_{n,i}\})$
- $2.n - 3$ East-West diagonals: $set(D_1, \{x_{1,2}, x_{2,1}\})$, $set(D_2, \{x_{1,3}, x_{2,2}, x_{3,1}\})$, $\ldots$, $set(D_{2.n-3}, \{x_{n-1,n}, x_{n,n-1}\})$
- $2.n - 3$ West-East diagonals: $set(D_{2.n-2}, \{x_{n-1,2}, x_{n,1}\})$, $\ldots$, $set(D_{4.n-6}, \{x_{1,n-1}, x_{2,n}\})$
- the set of n queens: $set(Q, \{x_{1,1}, \ldots, x_{n,n}\})$
- the n queens: $\forall i \in [1..n], Element(q(i), \{x_{1,1}, \ldots, x_{n,n}\})$

The constraints are:

- Q is of size n: $|Q| = n$
- the n queens are in Q: $\forall i \in [1..n], q(i) \in Q$
- queen i is on row i: $\forall i \in [1..n], q_i \in R_i$
- one and only one queen per column: $\forall i \in [1..n], set(CQ_i, \{x_{1,1}, \ldots, x_{n,n}\})$, $CQ_i = C_i \cap Q, |CQ_i| = 1$
- at most one queen per diagonal: $\forall i \in [1..4.n - 6], set(DQ_i, \{x_{1,1}, \ldots, x_{n,n}\})$, $DQ_i = D_i \cap Q, |DQ_i| < 2$

Note that the support of each CQ_i (resp. DQ_i) could have been set to the support of C_i (resp.D_i). However, one does not have to care about this when modeling since the $\Rightarrow_{red}$ rules will reduce these supports. The solutions are contained in Q: each element of Q is a queen, i.e., a cell of the board.

nqQ: Model with the queens as universe. Since the encoding is very correlated to the size of the support, we propose another model where the universe is much smaller, i.e., the set of n queens:

- Universe: $\mathcal{Q} = \{q_1, \ldots, q_n\}$, i.e., the n queens to be placed on a $n \times n$ board;
- Rows: $\forall i \in [1..n], set(R_i, \{q_i\})$; each row i is over queen i;
- the set of queens and columns are defined as above, but over the support $\mathcal{Q}$;
- each cell $C_{i,j}$ is defined as the intersection of row R_i and column C_j: $\forall i, j \in [1..n], set(C_{i,j}, \mathcal{Q}), C_{i,j} = R_i \cap C_j$;
- the $4.n - 6$ diagonals are defined by unions of cells: $set(D_1, \mathcal{Q})$, $D_1 = C_{1,2} \cup C_{2,1}$, $set(D_2, \mathcal{Q})$, $D_2 = C_{1,3} \cup C_{2,2} \cup C_{3,1}$, $\ldots$
- Q is of size n: $|Q| = n$
- to enforce one queen per row: $\forall i \in [1..n],\ |R_i| = 1$;
- one and only one queen per column: $\forall i \in [1..n],\ |C_i| = 1$;
- a different queen on each column: $Q = \bigcup_{i=1}^{n} C_i$
  (or, $\forall i, j \in [1..n],\ set(CC_{i,j}, \mathcal{Q}), CC_{i,j} = C_i \cap C_j, |CC_{i,j}| = 0$);

– atmost one queen per diagonal: $\forall i \in [1..4.n-6], |D_i| < 2$.

Interpretation of the results: if cell $C_{i,j} = \{q_k\}$, then queen q_k is in $i \times j$, else $C_{i,j} = \emptyset$ and there is no queen in $i \times j$.

As said before, our goal is not to compete with arithmetic solvers or set solvers, but to be able to declaratively, expressively, and error-prone model problems into SAT. Table **??** presents the results for the two models (model with the board as universe and model with the queens as universe). Column "q" represents the queens number and others columns represent the number of variables (var) and clauses (cl) for the generated SAT instance, the encoding time (time $\Leftrightarrow_{enc}$), the reduction time (time $\Rightarrow_{red}$) and the solving time by the Minisat solver [**?**] (minisat). When have limited the running time to 600 seconds for each combination of processes. No result is written if this value is reached. When only the Minisat column is empty this means that the instance exceed the memory size (4GB).

We can observe that the reduction rules $\Rightarrow_{red}$ permit to significantly reduce the size of the SAT instances. Thereby, instances which are unsolvable (due to the size) before reduction are now solved by MiniSat (q=30 for model nqB and q=120 for model nqQ). This result shows that contrary to some reduction rules such as breaking symmetry [**?**], our reduction rules do not make the search more difficult. Finally we can also note that the cumulative running time (encoding+resolution or reduction+encoding+resolution) is better when reduction is applied: always for the nqB model and from q=30 for the nqQ model.

We have tried our technique on various problems (e.g., Social Golfer Problem [**?**], Sudoku, Car-sequencing) and the SAT instances which are automatically generated have a complexity similar to the complexity of improved hand-written SAT formulations, and their solving with a SAT solver (in our case Minisat) is efficient compared to other SAT approaches.

6 Conclusion

We have presented a technique for encoding set constraints into SAT: the modeling process is achieved using some very declarative and expressive set constraints; they are then reduce by our $\Rightarrow_{red}$ rules before being automatically converted ($\Leftrightarrow_{enc}$) into SAT variables and clauses. We have illustrated our approach on the nqueen problem and shown some good results with the application of reduction and encoding rules. The advantages of our technique are the following:

– the modeling process is simple, declarative, expressive, and readable. Moreover, it is solver independent and independent from CSP or SAT solvers;
– the technique is less error-prone than hand-written 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.

Table 1. Experimental results

q	Model with the Board as universe								Model with the queens as universe							
	$\Leftrightarrow_{enc}$				$\Rightarrow_{red}+\Leftrightarrow_{enc}$				$\Leftrightarrow_{enc}$				$\Rightarrow_{red}+\Leftrightarrow_{enc}$			
	var	cl	time $\Leftrightarrow_{enc}$	minisat	time $\Rightarrow_{red}$	var	cl	time $\Leftrightarrow_{enc}$	minisat	var	cl	time $\Leftrightarrow_{enc}$	minisat	time $\Rightarrow_{red}$	var	cl	time $\Leftrightarrow_{enc}$	minisat
5	3 696	22 749	0,07	0,01	0,02	564	2 055	0,01	0,00	475	1 486	0,01	0,00	0,03	267	796	0,01	0,00
10	42 956	635 874	2,19	0,38	0,06	2 876	18 654	0,09	0,03	3 000	11 166	0,07	0,00	0,05	1 332	4 936	0,03	0,01
15	170 611	4 652 164	16,64	2,94	0,21	7 243	75 413	0,25	0,13	8 585	35 566	0,16	0,02	0,19	3 347	14 476	0,07	0,01
20	449 116	19 336 154	56,90	18,07	0,42	13 924	212 850	0,68	0,65	18 300	81 326	0,37	0,05	0,34	6 428	31 398	0,13	0,04
25	942 141	58 637 309	183,67	88,12	0,75	22 977	486 081	1,61	2,92	32 835	154 326	0,72	0,09	0,61	10 623	57 548	0,21	0,07
30	1 715 296	145 441 474	485,82		0,78	34 392	965 086	3,77	8,37	52 870	260 426	1,20	0,16	0,92	15 918	94 648	0,34	0,09
35					1,93	48 486	1 735 499	6,70	18,54	79 665	406 646	1,84	0,26	1,37	22 439	144 700	0,50	0,16
40					2,79	65 316	2 897 434	12,56	43,23	114 000	599 046	2,81	0,35	2,09	30 264	209 610	0,69	0,24
45					4,00	84 646	4 565 419	19,65	89,72	156 335	843 046	3,79	0,51	3,02	39 339	291 020	0,94	0,32
50					5,32	106 928	6 870 358	32,06	166,80	207 420	1 144 646	5,20	0,68	4,13	49 664	390 680	1,24	0,45
55					7,16	131 783	9 956 493	50,23	303,35	268 005	1 509 846	6,87	0,96	5,42	61 239	510 340	1,59	0,58
60					9,18	159 188	13 983 778	76,83	514,04	338 840	1 944 646	8,80	1,24	7,08	74 064	651 750	2,00	0,73
65										420 995	2 455 686	11,17	1,67	9,45	88 207	816 796	2,54	0,95
70										516 330	3 051 186	13,63	1,95	11,91	103 962	1 007 816	3,15	1,18
75										624 415	3 734 786	16,97	4,54	16,70	121 117	1 226 136	3,77	1,44
80										746 000	4 512 486	20,32	2,81	20,80	139 672	1 473 506	4,43	1,77
85										881 835	5 390 286	24,20	3,74	25,90	159 627	1 751 676	5,37	2,12
90										1 032 670	6 374 186	28,71	4,16	32,00	180 982	2 062 396	6,10	2,58
95										1 199 255	7 470 186	33,82	9,57	37,80	203 737	2 407 416	7,09	3,06
100										1 382 340	8 684 286	45,03	17,63	40,50	227 892	2 788 486	8,47	3,46
105										1 582 675	10 022 486	46,08	8,54	52,00	253 447	3 207 356	9,55	4,22
110										1 801 010	11 490 786	53,00	142,33	61,00	280 402	3 665 776	11,15	4,68
115										2 038 095	13 095 186	60,13	61,73	70,80	308 757	4 165 496	12,72	5,46
120										2 294 680	14 841 686	69,36		82,00	338 512	4 708 266	14,39	6,12
150										4 302 080	28 673 566	132,96		185,00	550 732	8 975 516	26,40	12,98
200										9 735 680	67 150 566	316,90		525,00				

In the future, we plan to use our set constraints encoding for formalizing finite domain variables. We also plan to combine set constraints with arithmetic constraints, and we want to define the corresponding combining SAT encoding. To this end, we will need to add some new constraints and to complete our $\Leftrightarrow_{enc}$ and $\Rightarrow_{red}$ rules.

References

1. Choco. http://www.emn.fr/z-info/choco-solver/.
2. K. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
3. Fahiem Bacchus. Gac via unit propagation. In *Proc. of CP 2007*, volume 4741 of *LNCS*, pages 133–147. Springer, 2007.
4. O. Bailleux and Y. Boufkhad. Efficient cnf encoding of Boolean cardinality constraints. In *Proc. of CP 2003*, volume 2833 of *LNCS*, pages 108–122. Springer, 2003.
5. Christian Bessière, Emmanuel Hebrard, and Toby Walsh. Local consistencies in sat. In *Selected Revised Papers of SAT 2003.*, volume 2919 of *LNCS*, pages 299–314. Springer, 2004.
6. Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proc. of SAT 2003*, volume 2919, pages 502–518, 2003.
7. T. Früwirth. *Constraint Handling Rules*. Cambridge University Press, 2009.
8. Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, 1979.
9. Ian Gent and Ines Lynce. A sat encoding for the social golfer problem. In *IJCAI'05 workshop on modelling and solving problems with constraints*, 2005.
10. Carmen Gervet. Conjunto: Constraint propagation over set constraints with finite set domain variables. In *Proc. of ICLP'94*, page 733. MIT Press, 1994.
11. F. Lardeux, E. Monfroy, B. Crawford, and R. Soto. Set constraint model and automated encoding into sat: Application to the social golfer problem. submitted to Annals of Operations Research.
12. Frédéric Lardeux, Eric Monfroy, Frédéric Saubion, Broderick Crawford, and Carlos Castro. Sat encoding and csp reduction for interconnected alldiff constraints. In *Proc. of MICAI 2009*, volume 5845 of *LNCS*, pages 360–371, 2009.
13. Bruno Legeard and Emmanuel Legros. Short overview of the clps system. In *Proc. of PLILP'91*, volume 528 of *LNCS*, pages 431–433. Springer, 1991.
14. A. Mackworth. *Encyclopedia on Artificial Intelligence*, chapter Constraint Satisfaction. John Wiley, 1987.
15. E. Monfroy. A coordination-based chaotic iteration algorithm for constraint propagation. In *Proc of ACM SAC'2000 (1)*, pages 262–269. ACM, 2000.
16. E. Monfroy, F. Saubion, and T. Lambert. On hybridization of local search and constraint propagation. In *Proc. of ICLP 2004*, volume 3132 of *LNCS*, pages 299–313. Springer, 2004.
17. Steven Prestwich and Andrea Roli. Symmetry breaking and local search spaces. volume 3524 of *Lecture Notes in Computer Science*, pages 273–287. Springer Berlin Heidelberg, 2005.
18. F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.
19. Markus Triska and Nysret Musliu. An improved sat formulation for the social golfer problem. *Annals of Operations Research*, 194(1):427–438, 2012.