Extending Multiple-Valued Clausal Forms with Linear Integer Arithmetic*

Carlos Ansótegui
Universitat de Lleida
Lleida, Spain

Miquel Bofill
Universitat de Girona
Girona, Spain

Felip Manyà
IIIA, CSIC
Bellaterra, Spain

Mateu Villaret
Universitat de Girona
Girona, Spain

Abstract

We extend the language of signed many-valued clausal forms with linear integer arithmetic constraints. In this way, we get a simple modeling language in which a wide range of practical combinatorial problems admit compact and natural encodings. We then define efficient translations from our language into the SAT and SMT formalism, and propose to use SAT and SMT solvers for finding solutions.

1 Introduction

Devising efficient and effective techniques for solving challenging combinatorial problems is a very active research topic in the Constraint Programming (CP) and Satisfiability Testing (SAT) research communities, and even solver competitions are regularly held as co-located events of the conferences in these fields. Roughly speaking, CP counts with rich modeling languages, and devoted algorithms for global constraints. SAT counts with a simple standard modeling language, and fully automatic (no tuning is needed) solvers that are highly competitive on real-word problems due to the incorporation of powerful techniques such as watched literals, clause learning, non-chronological backtracking and dynamic activity-based variable selection heuristics.

In this paper we present a new problem solving approach, based on multiple-valued logics, that bridges the gap between CP and SAT, and takes into account the most recently developed SAT technology. Our modeling language extends the signed multiple-valued clausal forms with linear integer arithmetic (LIA) constraints. Actually, our language may be seen as the multiple-valued counterpart of the Pseudo-Boolean formalism in the Boolean setting. This way, a wide range of combinatorial problems are more naturally and compactly encoded than just using Boolean/multiple-valued clausal forms, and at the same time take advantage of the best technology that SAT and Satisfiability Modulo Theory (SMT) solvers incorporate. A crucial point of our approach is that our modeling language is very simple, and can be efficiently translated into the mentioned SAT-based formalisms, as well as to many-valued SAT solvers and the solvers of the CP community. In contrast to more expressive languages, the problem of finding a solution remains decidable.

A suitable option would be to implement a solver for our language on top of a highly efficient many-valued SAT solver incorporating the recently developed SAT technology. This way, the structural information of the domain could be exploited in both the branching heuristics and the conflict-based clause learning module, and stronger forms of consistency could be enforced. Unfortunately, duplicating all the effort put in contemporary SAT solvers would be very costly. Therefore, we believe that a good alternative to build a solver with relatively low cost is to use efficient and effective encodings from our language into the SAT and SMT formalisms, and then use, depending on the structure of the instance to be solved, either a SAT solver or an SMT solver for finding solutions. Hence, in this paper, we first formally define the new language and then efficient translations from our language into the formalism used by SAT and SMT solvers. To the best of our knowledge, this is the first time that a link between SMT and signed clausal forms has been established, as well as that LIA constraints have been incorporated into the multiple-valued setting.

We would like to mention two recent works of the CP community that, together with the existing work in our community, have inspired the contributions of the present paper. On the one hand, a recent work [6] proposes to use many-valued clausal forms as a language for solving CP programs. Among other advantages, the authors state that the conflict-based clause learning of multiple-valued clausal forms provides a natural way of incorporating conflict-based clause learning in the CP framework when dealing with finite domains. On the other hand, one of the best performing

*Research partially supported by the Generalitat de Catalunya under grant 2009-SGR-1434, the *Ministerio de Ciencia e Innovación* research projects CONSOLIDER CSD2007-0022, INGENIO 2010, Acción Integrada HA2008-0017, TIN2008-04547, TIN2009-14704-C03-01, and TIN2010-20967-C04-01/03, and the *Secretaría General de Universidades del Ministerio de Educación: Programa Nacional de Movilidad de Recursos Humanos.*

solvers in the CP competitions is Sugar [9]. This solver translates CP problems into SAT but instead of using translations based on literals of the form $x = i$, it uses literals of the form $x \geq i$, which have been widely used in multiple-valued logic and are known as regular literals [4, 5]. Surprisingly, the authors of Sugar ignored all the existing work on regular literals developed in our community.

This paper is structured as follows. Section 2 defines the modeling language. Section 3 defines efficient mappings from our formalism into the SAT and SMT formalisms. Section 4 presents the conclusions and future work.

2 Modeling Language

We first formally define the syntax and semantics of the many-valued clausal forms considered in our work, and introduce Linear Integer Arithmetic (LIA) constraints. Then, we define our modeling language.

Definition 1. A truth value set, or domain, N is a non-empty finite set $\{i_1, i_2, \ldots, i_n\}$ where $n \in \mathbb{N}$. The cardinality of N is denoted by $|N|$. A total order $\leq$ is associated with N.

For the sake of simplicity, we assume that, unless otherwise stated, all the propositional variables have the same domain. Extending our results to the case in which every variable has a different domain is straightforward. We also assume that domains are of the form $\{1, 2, \ldots, |N|\}$.

Definition 2. A sign is a subset of the truth value set. A signed literal is an expression of the form $S{:}x$, where S is a sign and x is a propositional variable. The complement of a signed literal $S{:}x$, denoted by $\overline{S}{:}x$, is $(N \setminus S){:}x$. A signed clause is a disjunction of signed literals. A signed CNF formula is a conjunction of signed clauses.

Definition 3. Given a truth value set N and a value $i_k \in N$, a sign S is regular if it is either of the form $\{i \in N | i \geq i_k\}$ or of the form $N \setminus \{i \in N | i \geq i_k\}$. A signed literal is a regular literal if its sign is regular. A sign S is monosigned if it is either a singleton (i.e. it contains exactly one truth value) or the complement of a singleton. A signed literal is a monosigned literal if its sign is monosigned. A monosigned literal is positive if it is identical to $\{i_k\}{:}x$, and is negative if it is identical to $\{\overline{i_k}\}{:}x$.

In the sequel, we represent regular literals of the form $\{i \in N | i \geq i_k\}{:}x$ by $x \geq i_k$, regular literals of the form $N \setminus \{i \in N | i \geq i_k\}{:}x$ by $\neg(x \geq i_k)$ or $x \leq i_k - 1$, positive monosigned literals of the form $\{i_k\}{:}x$ by $x = i_k$, negative monosigned literals of the form $\{\overline{i}\}{:}x$ by $\neg(x = i_k)$. Without loss of generality, all our signed CNF formulas contain only regular and monosigned literals.

Definition 4. An assignment is a mapping that assigns to every propositional variable an element of the truth value set. An assignment I satisfies a signed literal $S{:}x$ iff $I(x) \in S$, satisfies a signed clause C iff it satisfies at least one of the signed literals in C, and satisfies a signed CNF formula Γ iff it satisfies all the clauses in Γ. A signed CNF formula is satisfiable iff it is satisfied by at least one assignment; otherwise it is unsatisfiable.

Definition 5. A linear integer arithmetic (LIA) constraint is an expression of the form $\sum_{i=1}^{m} a_i x_i \otimes c$, where $\otimes \in \{\leq, \geq, =\}$, c is an integer, a_i is a non-zero integer, and x_i is a propositional variable for each i ($1 \leq i \leq m$). An assignment satisfies a LIA constraint if the inequality or equality $\otimes$ holds when the propositional variables are instantiated by the values of the assignment.

Definition 6. A multiple-valued formula with LIA constraints (MVL-formula) is a set of signed clauses and LIA constraints. An assignment satisfies an MVL-formula if it satisfies both all its signed clauses and all its LIA constraints.

Our new modeling language is the language of MVL-formulas. Notice that it may be seen as the multiple-valued counterpart of the Pseudo-Boolean formalism in the Boolean setting.

Example 1. A magic square of order n is an arrangement of the integers $1, 2, \ldots, n^2$ in an $n \times n$ matrix in such a way that the n numbers in all rows, all columns, and both diagonals sum to the magic constant $M = n(n^2 + 1)/2$. If we would like to find a magic square of order 3 in our formalism, we should find a satisfying assignment for the following MVL-formula over the domain $N = \{1, 2, \ldots, 9\}$:

$$
\begin{array}{c}
x_{11} = 1 \vee \ldots \vee x_{ij} = 1 \vee \ldots \vee x_{33} = 1 \\
x_{11} = 2 \vee \ldots \vee x_{ij} = 2 \vee \ldots \vee x_{33} = 2 \\
\vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
x_{11} = 9 \vee \ldots \vee x_{ij} = 9 \vee \ldots \vee x_{33} = 9 \\
x_{11} + x_{12} + x_{13} = 15 \\
x_{21} + x_{22} + x_{23} = 15 \\
x_{31} + x_{32} + x_{33} = 15 \\
x_{11} + x_{21} + x_{31} = 15 \\
x_{12} + x_{22} + x_{32} = 15 \\
x_{13} + x_{23} + x_{33} = 15 \\
x_{11} + x_{22} + x_{33} = 15 \\
x_{13} + x_{22} + x_{31} = 15
\end{array}
$$

Variable x_{ij} denotes the value of row i and column j, and $1 \leq i, j \leq 3$.

Proposition 7. *The satisfiability problem for MVL-formulas is NP-complete.*

3 Mappings

3.1 Mapping MVL-formulas into SAT

We first define the translation of a LIA constraint into an equisatisfiable Signed CNF formula, and then the translation of a Signed CNF formula into an equisatisfiable Boolean CNF formula. In this way, solving an MVL-formula with SAT amounts to replacing their LIA constraints with their Signed SAT encoding, translating the derived Signed SAT instance into SAT, and feeding the resulting SAT instance to a SAT solver.

3.1.1 LIA Constraints into Signed SAT

We describe a novel Signed SAT encoding of LIA constraints, and propose to give a geometrical interpretation to encodings in order to produce more compact models. Observe that a LIA constraint $\sum_{i=1}^{m} a_i x_i \otimes c$ over a domain N, where $\otimes \in \{\leq, \geq, =\}$, is an m-ary constraint. Since $m \geq 2$, we will start by representing this constraint using the signed version of well-know direct encoding from CSP into SAT [7].

In the sequel we focus our attention on constraints of the form $\sum_{i=1}^{m} a_i x_i \leq c$, because $\sum_{i=1}^{m} a_i x_i \geq c$ is equivalent to $\sum_{i=1}^{m} -a_i x_i \leq -c$, and $\sum_{i=1}^{m} a_i x_i = c$ is equivalent to $\sum_{i=1}^{m} a_i x_i \leq c$ and $\sum_{i=1}^{m} a_i x_i \geq c$.

The idea behind the direct encoding is to represent as clauses the forbidden assignments (nogoods). In the simplest case, where the arity is 2, if we have a LIA constraint $a_1 x_1 + a_2 x_2 \leq c$, we get the following encoding:

$$\bigwedge_{\substack{a_1 v_1 + a_2 v_2 > c \\ v_1, v_2 \in N}} \neg(x_1 = v_1 \wedge x_2 = v_2)$$

For example, the constraint $x_1 + 2x_2 \leq 6$ over the domain $N = \{1, 2, 3, 4\}$ is:

$$\begin{array}{ll} \neg(x_1 = 1 \wedge x_2 = 3) & \neg(x_1 = 1 \wedge x_2 = 4) \\ \neg(x_1 = 2 \wedge x_2 = 3) & \neg(x_1 = 2 \wedge x_2 = 4) \\ \neg(x_1 = 3 \wedge x_2 = 2) & \neg(x_1 = 3 \wedge x_2 = 3) \\ \neg(x_1 = 3 \wedge x_2 = 4) & \neg(x_1 = 4 \wedge x_2 = 2) \\ \neg(x_1 = 4 \wedge x_2 = 3) & \neg(x_1 = 4 \wedge x_2 = 4) \end{array}$$

If we represent the space of all the possible assignments of x_1 and x_2 in a matrix, and mark with (v_i, v_j) the nogoods and with "$-$" the goods, we get:

	$x_1 = 1$	$x_1 = 2$	$x_1 = 3$	$x_1 = 4$
$x_2 = 4$	$(1,4)$	$(2,4)$	$(3,4)$	$(4,4)$
$x_2 = 3$	$(\mathbf{1},\mathbf{3})$	$(\mathbf{2},\mathbf{3})$	$(3,3)$	$(4,3)$
$x_2 = 2$	$-$	$-$	$(\mathbf{3},\mathbf{2})$	$(\mathbf{4},\mathbf{2})$
$x_2 = 1$	$-$	$-$	$-$	$-$

Thinking geometrically, we have that the possible assignments to x_1 and x_2 correspond to a square, and the arithmetic constraint corresponds to a straight line that divides the square into two regions: one containing all the goods and another containing all the nogoods. Hence, the direct encoding amounts to explicitly represent all the points of the square that are in the nogood region. However, we can get more compact representations if a clause encodes an area of the nogood region containing several nogoods instead of encoding exactly one nogood. This is the idea behind our proposal to defining a compact encoding of LIA constraints using signed literals. Even when in this paper we focus on LIA constraints, this geometrical interpretation may lead to more compact Signed SAT and SAT encodings on a wide range of constraints, and provides evidence of the advantages of using signed literals for modeling combinatorial problems.

A regular version of the direct encoding for a LIA constraint of the form $a_1 x_1 + a_2 x_2 \leq c$, where $a_1 > 0$ and $a_2 > 0$, was defined in [9] as follows: [1]

$$\bigwedge_{\substack{b_1 + b_2 = c + 1 \\ \left\lceil \frac{b_1}{a_1} \right\rceil, \left\lceil \frac{b_2}{a_2} \right\rceil \in N}} \neg(x_1 \geq \left\lceil \frac{b_1}{a_1} \right\rceil \wedge x_2 \geq \left\lceil \frac{b_2}{a_2} \right\rceil)$$

In general, for a LIA constraint of the form $a_1 x_1 + \cdots + a_m x_m \leq c$, the encoding becomes:

$$\bigwedge_{\substack{\sum_{i=1}^{m} b_i = c + 1 \\ \left\lceil \frac{b_i}{a_i} \right\rceil \in N}} \neg(x_1 \geq \left\lceil \frac{b_1}{a_1} \right\rceil \wedge \cdots \wedge x_m \geq \left\lceil \frac{b_m}{a_m} \right\rceil)$$

In our example, the generated clauses are:

$$\begin{array}{ll} \neg(x_1 \geq 1 \wedge x_2 \geq 3) & \neg(x_1 \geq 2 \wedge x_2 \geq 3) \\ \neg(x_1 \geq 3 \wedge x_2 \geq 2) & \neg(x_1 \geq 4 \wedge x_2 \geq 2) \end{array}$$

Thanks to the regular signs, this regular direct encoding produces fewer clauses than the standard direct encoding with monosigned signs.

Let us see the geometrical interpretation of the clauses of the previous regular direct encoding: the clause $\neg(x_1 \geq 1 \wedge x_2 \geq 3)$ represents the area containing the nogoods $(1,4), (2,4), (3,4), (4,4), (1,3), (2,3), (3,3), (4,3)$; the clause $\neg(x_1 \geq 2 \wedge x_2 \geq 3)$ represents the area containing $(2,4), (3,4), (4,4), (2,3), (3,3), (4,3)$, the clause $\neg(x_1 \geq 3 \wedge x_2 \geq 2)$ represents the area containing $(3,4), (3,3), (3,2), (4,4), (4,3), (4,2)$, and the clause $\neg(x_1 \geq 4 \wedge$

[1] The authors of [9] do not describe their encoding as a direct encoding. We present here a reformulation, based on our geometrical interpretation, which shows that their encoding is, in fact, a regular direct encoding. If a_1 (a_2) is negative, then $x_1 \geq \left\lceil \frac{b_1}{a_1} \right\rceil$ ($x_2 \geq \left\lceil \frac{b_2}{a_2} \right\rceil$) should be replaced with $x_1 \leq \left\lfloor \frac{b_1}{a_1} \right\rfloor$ ($x_2 \leq \left\lfloor \frac{b_2}{a_2} \right\rfloor$).

$x_2 \geq 2$) represents the area containing $(4,4),(4,3),(4,2)$. Observe that this regular direct encoding defines areas of the nogood region whose union covers all the nogoods. However, there is a considerable overlap among these areas. So, we propose a signed encoding (not necessarily regular) in which the clauses encode areas of the nogood region and their union covers all the nogoods but there is no overlap. To this end, we start by deriving the above regular encoding. Then, we remove redundant clauses. In our example, we remove the clauses $\neg(x_1 \geq 2 \wedge x_2 \geq 3)$ and $\neg(x_1 \geq 4 \wedge x_2 \geq 2)$ because they are proper subsets of one of the remaining clauses. Nevertheless, after removing redundant clauses, there is yet overlap: the clauses $\neg(x_1 \geq 1 \wedge x_2 \geq 3)$ and $\neg(x_1 \geq 3 \wedge x_2 \geq 2)$ overlap in the area formed by $(3,4),(4,4),(3,3),(4,3)$. To overcome this drawback, we will use interval signs. Interval signs allow to remove overlapping areas once redundant clauses have been removed. In our example, the encoding would be formed by two clauses: $\neg([1,2] : x_1 \wedge x_2 \geq 3)$ and $\neg([3,4] : x_1 \wedge x_2 \geq 2)$. In order to have a complete encoding in our language we may transform the interval signed literals into a disjunction of monosigned literals. Otherwise, we could incorporate interval signed literals in our language. In this case, we just have to add, for every occurring literal $[a,b] : x$, the clausal form of $[a,b] : x \leftrightarrow x \geq a \wedge \neg(x \geq b+1)$.

We have so far discussed the new encoding for binary constraints and its geometrical interpretation, but our approach may also be applied to non-binary constraints. For instance, for ternary constraints, the space of all possible interpretations is represented by a cube, and a LIA constraint of the form $a_1x_1 + a_2x_2 + a_3x_3 \leq c$ is a plane that intersects with that cube. To obtain a compact encoding, we propose to use parallelepipeds represented by clauses of the form $\neg([i_1,j_1] : x_1 \wedge [i_2,j_2] : x_2 \wedge x_3 \geq k)$[2], which are parallelepipeds with width $j_1 - i_1$, height $j_2 - i_2$, and length $|N| - k$. All these parallelepipeds may have different volume, and allow to cover the nogood region without overlap. In the general case, we will have expressions of the form $\neg([i_1,j_1] : x_1 \wedge \cdots \wedge [i_{m-1},j_{m-1}] : x_{m-1} \wedge x_m \geq k)$.

3.1.2 Signed SAT into SAT

Translating signed CNF formulas into satisfiability equivalent Boolean CNF formulas has been proposed in [1, 3]. It amounts to interpreting signed literals as Boolean literals, and adding, for each propositional variable, the following Boolean clauses:[3]

$$
\begin{array}{ll}
x \geq |N| \rightarrow x{\geq}|N|{-}1 & x = 1 \leftrightarrow \neg x{\geq}2 \\
x{\geq}|N|{-}1 \rightarrow x{\geq}|N|{-}2 & x = 2 \leftrightarrow x{\geq}2 \wedge \neg(x{\geq}3) \\
\cdots\cdots\cdots\cdots & \cdots\cdots\cdots\cdots \\
x{\geq}3 \rightarrow x{\geq}2 & x = i \leftrightarrow x{\geq}i \wedge \neg(x{\geq}i+1) \\
x{\geq}2 \rightarrow x{\geq}1 & \cdots\cdots\cdots\cdots \\
 & x = |N|{-}1 \leftrightarrow x{\geq}|N|{-}1 \wedge \neg(x{\geq}|N|) \\
 & x = |N| \leftrightarrow x{\geq}|N|
\end{array}
\tag{1}
$$

The clauses on the left encode the order relation while the clauses on the right link monosigned and regular literals. It turns out that the derived Boolean CNF formula and the input signed CNF formula are equisatisfiable.

3.2 Mapping MVL-formulas into SMT

An SMT instance is a generalization of a Boolean formula in which some propositional variables have been replaced by predicates with predefined interpretations from background theories. For example, a formula can contain clauses like, e.g., $p \vee q \vee (x+2 \leq y) \vee (x > y+z)$, where p and q are Boolean variables, and x, y and z are integer variables. Predicates over non-Boolean variables, such as linear integer inequalities, are evaluated according to the rules of a background theory. Examples of theories include linear real or integer arithmetic, arrays, bit vectors, etc., or combinations of them.

Formally speaking, a *theory* is a set of first-order formulas closed under logical consequence. The SMT problem for a theory T is: given a first-order formula F, determine whether there is a model of $T \cup \{F\}$.

Although an SMT instance can be solved by encoding it into an equisatisfiable SAT instance and feeding it to a SAT solver, currently most successful SMT solvers are based on the integration of a SAT solver and a T-solver, that is, a decision procedure for the given theory T. In this so-called lazy approach, while the SAT solver is in charge of the Boolean component of reasoning, the T-solver deals with sets of atomic constraints in T. The main idea is that the T-solver analyzes the partial model that the SAT solver is building, and warns it about conflicts with theory T (T-inconsistency). In this way, we are hopefully getting the best of both worlds: in particular, the efficiency of the SAT solver for the Boolean reasoning and the efficiency of special-purpose algorithms inside the T-solver for the theory reasoning. See [8] for a survey on this approach.

Among the theories considered in the SMT library [2] we are interested in integer numbers and fixed size bit vectors, and more specifically in the logics:

- QF_LIA: quantifier-free *linear integer arithmetic*. In essence, closed quantifier-free formulas with Boolean

[2]Depending on the type of intersection of the plane and the cube, the variables containing interval (regular) literals may be different.

[3]For the sake of clarity, in this paper we sometimes use $a \rightarrow b$ and $\neg(a \wedge b)$ instead of their clausal form: $\neg a \vee b$ and $\neg a \vee \neg b$, respectively.

combinations of inequalities between linear polynomials over integer variables.

- QF_IDL: difference logic over the integers. A fragment of QF_LIA where inequalities are restricted to have the form $x - y \otimes b$ being x and y integer variables, b an integer constant and $\otimes \in \{<, \leq, >, \geq, =, \neq\}$.

- QF_BV: closed quantifier-free formulas over bit vectors, with operations such as concatenation, extraction and the usual bitwise logical operations.

Next we describe how to translate an MVL-formula into SMT instances using the above logics. For QF_LIA the translation is direct, whereas for QF_IDL and QF_BV we first need to translate the MVL-formula into a signed CNF formula (as shown in Subsection 3.1).

3.2.1 Linear Integer Arithmetic

QF_LIA is a language more general than our MVL-formulas and therefore it captures such formulas quite naturally.

Example 2. The SMT encoding of Example 1 in the SMT-LIB language v1.2 under QF_LIA is as follows:

```
(benchmark magic_seq
 :logic QF_LIA
 :extrafuns ((x11 Int) (x12 Int) (x13 Int)
             (x21 Int) (x22 Int) (x23 Int)
             (x31 Int) (x32 Int) (x33 Int))
 :formula
 (and
  (and (>= x11 1) (<= x11 9)
                ...
       (>= x33 1) (<= x33 9))
  (and (or (= x11 1) ... (= x33 1))
                ...
       (or (= x11 9) ... (= x33 9)))
  (and (= 15 (+ x11 x12 x13))
       (= 15 (+ x21 x22 x23))
       (= 15 (+ x31 x32 x33))
       (= 15 (+ x11 x21 x31))
       (= 15 (+ x12 x22 x32))
       (= 15 (+ x13 x23 x33))
       (= 15 (+ x11 x22 x33))
       (= 15 (+ x13 x22 x31)))
 )
)
```

In the above example we can find three sections. The first one determines the logic to be used. The second one declares the variables occurring in the formula. The third one contains the formula, which must not be necessarily in clausal form. Notice that the first part of the formula constraints the domains of the variables.

3.2.2 Integer Difference Logic

Given a signed CNF formula, we translate regular literals of the form $x \geq a$ into the difference logic atom $(\geq \texttt{x a})$, and monosigned literals of the form $x = a$ into the difference logic atom $(= \texttt{x a})$. Apart from this translations, we must define that each variable x has domain N: $(\texttt{and } (\geq \texttt{x 1}) \ (\leq \texttt{x } |\texttt{N}|))$.

3.2.3 Bit Vectors

We now describe how a signed CNF formula can be encoded as an equisatisfiable QF_BV formula. Given a signed CNF formula F, for each distinct propositional variable x and each distinct signed literal $S_i : x$ with x, we create: (i) a Boolean variable x_i; (ii) a bit vector variable b_{x_i} whose length is the cardinality $|N|$ of the domain; and (iii) a bit vector constant c_{x_i} of the same length whose j-th bit is 1 iff $j \in S_i$.

Next, we define:

Boolean mapping of F as the formula resulting from replacing each signed literal of the form $S_i{:}x$ in F with x_i.

Bit vector linking of a signed literal $S_i{:}x$ as the formula:

$$x_i \rightarrow (b_{x_i} = c_{x_i}) \wedge \neg x_i \rightarrow (b_{x_i} = (\mathsf{bvnot}\ c_{x_i}))$$

Non-emptiness of a propositional variable x in F as the formula:

$$(\mathsf{bvand}\ b_{x_1} \ldots b_{x_n}) \neq 0$$

where $S_1{:}x, \ldots, S_n{:}x$ are all the signed literals with x in F.

Then, the conjunction of the Boolean mapping of F, the bit vector linking of every signed literal $S_i{:}x$ in F, and the non-emptiness formula of every propositional variable x in F, gives us a QF_BV formula which is equisatisfiable to F.

Notice that our formulation is valid not only for monosigned and regular literals, but for all signed literals $S_i{:}x$ in general, since the bit vector constants c_{x_i} explicitly represent the sign S_i.

Recall from Definition 4 that an assignment I maps every propositional variable x to an element of the domain, and it satisfies a signed literal $S{:}x$ iff $I(x) \in S$. Therefore, the QF_BV formula obtained from a signed formula F is equisatisfiable to F since, for each $S_i{:}x$ in F:

- The j-th bit of c_{x_i} is 1 iff $j \in S_i$.

- The vector linking formulas guarantee that the $I(x)$-th bit of b_{x_i} is set to 1. Notice that $S_i{:}x$ has been replaced by x_i and, when x_i is true, b_{x_i} is equal to c_{x_i} (the bit vector representing S_i) and, when x_i is false, b_{x_i} is equal to its complement.

The non-emptiness constraint, for each variable x, ensures consistency of $I(x)$ with $I'(x_i)$ for all signed literals $S_i{:}x$, where I' is a Boolean assignment. Since bvand is a bitwise AND on bitvectors, we are enforcing that, at least for one k in the domain (including $I(x)$), the k-th bit of all b_{x_i} is 1 and, hence, consistency.

Example 3. Let F be a signed CNF formula of the form $\{\{1,2\}{:}x \vee D_1, \{1,3\}{:}x \vee D_2, \ldots\}$, where D_1, D_2 are disjunctions of signed literals. Assuming that $\{1,2\}{:}x$ and $\{1,3\}{:}x$ are the only signed literals with x in F, and that the domain is $\{1,\ldots,8\}$, the resulting QF_BV formula is:

```
(benchmark bv.smt
:logic QF_BV
:extrapreds ((x1) (x2) ...)
:extrafuns ((b_x1 BitVec[8])
            (b_x2 BitVec[8])
            ...
           )
:formula
 (and
  (or x1 ...)
  (or x2 ...)
  ...
  (implies      x1  (= b_x1 bvbin00000011))
  (implies (not x1) (= b_x1 bvbin11111100))
  (implies      x2  (= b_x2 bvbin00000101))
  (implies (not x2) (= b_x2 bvbin11111010))
  (not (= (bvand b_x1 b_x2) bvbin00000000))
 )
)
```

4 Conclusions

We have presented a new multiple-valued modeling language that extends signed CNF formulas with LIA constraints. Our language allows to model in a natural and compact way a wide range of practical combinatorial problems.

We have defined efficient mappings from MVL-formulas to both SAT and SMT. We claim that it is better to model the problems to be solved in our language and then translate them to SAT or SMT. Notice that if we perform a direct translation from SAT to SMT, we have to treat the SMT variables as Boolean variables because the domain information is lost in the SAT encoding.

Regarding the question of whether it is better to use either a SAT solver or an SMT solver, we believe that this depends on the particular structure of the problem to be solved. Fortunately, our simple modeling language allows the translation to both SAT and SMT. We also plan to provide direct translations into CP solvers.

It is worth noticing that, for the first time, we extended signed CNF formulas with LIA constraints, and defined novel encodings from this new language into SAT. Furthermore, we have introduced a geometrical interpretation of the direct encoding that allows one to produce more compact and natural encodings of LIA constraints when signed literals are used. We plan to apply this approach to other constraints, and empirically evaluate the impact of eliminating redundancies. We have also established, for the first time, a link between signed clausal forms and SMT formalisms, as well as defined efficient encodings from our language into SMT. On the one hand, we have shown that SMT allows one to deal with signed CNF formulas and LIA constraints using a high-level language. On the other hand, we have defined a trickier encoding of signed CNF formulas based on bit vectors that leads to a new approach to solving multiple-valued SAT problems. The next step is to design and conduct an empirical evaluation of the contributions of the paper.

References

[1] C. Ansótegui and F. Manyà. Mapping problems with finite-domain variables into problems with Boolean variables. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (Revised Selected Papers), SAT-2004, Vancouver, Canada*, pages 1–15. Springer LNCS 3542, 2004.

[2] C. Barrett, A. Stump, and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`, 2010.

[3] R. Béjar, R. Hähnle, and F. Manyà. A modular reduction of regular logic to classical logic. In *Proceedings, 31st International Symposium on Multiple-Valued Logics (ISMVL), Warsaw, Poland*, pages 221–226. IEEE CS Press, Los Alamitos, 2001.

[4] R. Hähnle. Uniform notation of tableau rules for multiple-valued logics. In *Proc. International Symposium on Multiple-Valued Logics, ISMVL'91, Victoria, B.C., Canada*, pages 238–245. IEEE Press, Los Alamitos, 1991.

[5] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*, volume 10 of *International Series of Monographs in Computer Science*. Oxford University Press, 1994.

[6] S. Jain, E. O'Mahony, and M. Sellmann. A complete multi-valued SAT solver. In *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming, CP-2010, St. Andrews, Scotland*, pages 281–296. Springer LNCS 6308, 2010.

[7] S. D. Prestwich. CNF encodings. In A. Biere, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, pages 75–97. IOS Press, 2009.

[8] R. Sebastiani. Lazy Satisfiability Modulo Theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3(3-4):141–224, 2007.

[9] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(4):254–272, 2009.