SAT and SMT Technology for Many-Valued Logics

CARLOS ANSÓTEGUI[1], MIQUEL BOFILL[2], FELIP MANYÀ[3], MATEU VILLARET[2]

[1] *Departament d'Informàtica i Enginyeria Industrial, Universitat de Lleida, Spain*
[2] *Departament d'Informàtica, Matemàtica Aplicada i Estadística, Universitat de Girona, Spain*
[3] *Artificial Intelligence Research Institute (IIIA, CSIC), Spain*

We propose to use the SAT and SMT technology to deal with many-valued logics. Our approach is twofold. Firstly, we focus on finitely-valued logics, and extend the language of signed CNF formulas with linear integer arithmetic constraints. 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 formalisms, and propose to use SAT and SMT solvers for finding solutions.

Secondly, we show how we can use the SMT technology to build efficient automated theorem provers for infinitely-valued logics, taking Łukasiewicz infinitely-valued logic as a case study.

Key words: many-valued logics, Łukasiewicz infinitely-valued logic, SAT, SMT, linear integer arithmetic, linear real arithmetic, bit vectors, regular encodings

1 INTRODUCTION

A many-valued logic (also known as multi-valued or multiple-valued logic) is a propositional logic where the truth value set has more than two truth values,

and each connective has a specific semantics. Depending on the cardinality of the truth value set, we distinguish between finitely-valued logics and infinitely-valued logics.

In the present paper, with respect to finitely-valued logics, we focus on the logic of signed CNF formulas [9], which provide a suitable and effective framework for solving combinatorial problems [3, 11], as well as for efficiently mapping the satisfiability problem of any finitely-valued logic [7].

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. Even solver competitions are regularly held as co-located events of the major 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.

The first contribution of this work is a new problem solving approach, based on many-valued logics, that bridges the gap between CP and SAT, and takes into account the most recently developed SAT technology. Our modeling language extends signed CNF formulas with linear integer arithmetic (LIA) constraints. Actually, our language may be seen as the many-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/many-valued clausal forms, and at the same time take advantage of the best technology that SAT and Satisfiability Modulo Theories (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 the language used by the existing 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, for our first contribution, we start by formally defining the new language and then provide 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 many-valued logics has been established, as well as that LIA constraints have been incorporated into the many-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 first contribution of the present paper. On the one hand, a recent work [17] proposes to use many-valued clausal forms as a language for solving CP problems. Among other advantages, the authors state that the conflict-based clause learning of many-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 solvers in the CP competitions is Sugar [22]. 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 many-valued logic and are known as regular literals [14, 15].

As said above, the significance of signed CNF formulas is not limited to provide a suitable and effective problem solving framework. Actually, they originated in the area of automated theorem proving in many-valued logics, where it was shown that the satisfiability problem of any finitely-valued logic is polynomially reducible to the satisfiability problem of signed CNF formulas [7]. Unfortunately, signed CNF formulas are not well-suited for dealing with infinitely valued logics too. Therefore, it is not clear how infinitely-valued logics can take advantage from SAT solvers.

The second contribution of this work is to show how we can use the SMT technology to build efficient automated theorem provers for infinitely-valued logics, taking Łukasiewicz infinitely-valued logic as a case study. In particular, we compare our approach with the implementations of theorem provers based on symbolic calculi for Łukasiewicz logic presented in [20]. Our approach is quite natural and easy to extend to other infinitely-valued logics. Moreover, we have conducted an experimental investigation which confirms that this approach dramatically outperforms the existing approaches to our best knowledge.

The paper is structured as follows. Section 2 extends many-valued clausal forms with linear integer arithmetic. Sections 3 and 4 define efficient mappings from the new formalism into the SAT and SMT formalisms. Section 5 presents and SMT-based automated theorem prover for the infinitely-valued logic of Łucasiewicz. Section 6 presents the conclusions and future work.

A preliminary short version of the results in this paper was presented at the 41st IEEE International Symposium on Multiple-Valued Logic [1].

2 EXTENDING MANY-VALUED CLAUSAL FORMS WITH LINEAR INTEGER ARITHMETIC

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 $\{\bar{i}\}{:}x$ 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) expression is an expression of the form $\sum_{i=1}^{m} a_i x_i$, where a_i is a non-zero integer, and x_i is a propositional variable for each i ($1 \leq i \leq m$). Given a LIA expression e, we denote its lower bound by $l(e)$ and its upper bound by $u(e)$.*

Definition 6 *Let e be a LIA expression. Then a LIA constraint is an expression of the form $e \otimes c$, where $\otimes \in \{\leq, \geq, =\}$ and c is an integer. 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 7 *A many-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 many-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*

$N = \{1, 2, \ldots, 9\}$:

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

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

Proposition 1 *The satisfiability problem for MVL-formulas is NP-complete.*

This proposition follows from the fact that the SAT problem can be efficiently reduced to the satisfiability problem for MVL-formulas, and that a satisfying assignment can be verified in polynomial time.

3 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. 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 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 [5, 6, 19].

In the sequel we focus our attention on constraints of the form $\sum_{i=1}^{m} a_i x_i \leq c$, since $\sum_{i=1}^{m} a_i x_i \geq c$ 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 $a_1x_1 + a_2x_2 \leq c$, where $a_1 > 0$ and $a_2 > 0$, was defined in [22] as follows:

$$\bigwedge_{b_1+b_2=c-1} \left(x_1 \leq \left\lfloor \frac{b_1}{a_1} \right\rfloor \vee x_2 \leq \left\lfloor \frac{b_2}{a_2} \right\rfloor \right)$$

where b_1, b_2 range over $\mathbb{Z}$, satisfying $l(a_1x_1) - 1 \leq b_1 \leq u(a_1x_1)$ and $l(a_2x_2) - 1 \leq b_2 \leq u(a_2x_2)$.

The authors of [22] 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* :

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

In general, for a LIA constraint of the form $a_1x_1 + \cdots + a_mx_m \leq c$, the encoding becomes:

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

A proof of the correctness of the previous encoding is given below. Before we give an example. Following our example on the constraint $x_1 + 2x_2 \leq 6$ over the domain $N = \{1, 2, 3, 4\}$, the generated clauses would be:

$$\neg(x_1 \geq 1 \wedge x_2 \geq 3) \quad \neg(x_1 \geq 2 \wedge x_2 \geq 3)$$
$$\neg(x_1 \geq 3 \wedge x_2 \geq 2) \quad \neg(x_1 \geq 4 \wedge x_2 \geq 2)$$

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 $\neg(x_1 \geq 4 \wedge x_2 \geq 2)$ represents the area containing $(4,4), (4,3), (4,2)$. Figure 1 graphically illustrates this interpretation.

* If a_1 (a_2) is negative, then $x_1 \geq \lceil \frac{b_1}{a_1} \rceil$ ($x_2 \geq \lceil \frac{b_2}{a_2} \rceil$) must be replaced with $x_1 \leq \lfloor \frac{b_1}{a_1} \rfloor$ ($x_2 \leq \lfloor \frac{b_2}{a_2} \rfloor$).

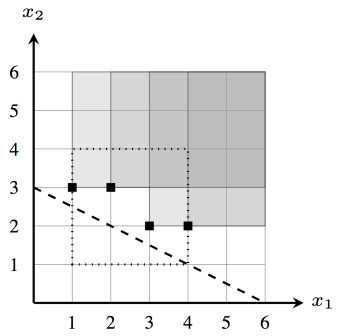

FIGURE 1
Geometrical interpretation for $x_1 + 2x_2 \leq 6$ over the domain $N = \{1, 2, 3, 4\}$.

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 possi-

ble 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 $\neg([i_1, j_1] : x_1 \wedge [i_2, j_2] : x_2 \wedge x_3 \geq k)$, 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 $\neg([i_1, j_1] : x_1 \wedge \cdots \wedge [i_{m-1}, j_{m-1}] : x_{m-1} \wedge x_m \geq k)$.

Now we proceed to prove the correctness of the proposed encoding. First of all, we recall the following Lemma from [22].

Lemma 1 ([22]) *For any LIA expressions e, f, and for any integer $c \geq l(e) + l(f)$, the following holds:*

$$e + f \leq c \quad \Longleftrightarrow \quad \bigwedge_{a+b=c-1} (e \leq a \vee f \leq b)$$

where the parameters a and b range over $\mathbb{Z}$, satisfying $l(e) - 1 \leq a \leq u(e)$ and $l(f) - 1 \leq b \leq u(f)$.

It is not difficult to see that the lemma also holds for a narrower range for a and b, namely $l(e) - 1 \leq a \leq u(e) - 1$ and $l(f) - 1 \leq b \leq u(f) - 1$. Roughly, if $a = u(e)$ then the clause $e \leq u(e) \vee f \leq b$ is trivially true (and similarly for b). However, the wider range is preferred by the authors of [22] to simplify the discussion.

Corollary 1 *For any LIA expressions e, f, and for any integer $c \geq l(e) + l(f)$, the following holds:*

$$e + f \leq c \quad \Longleftrightarrow \quad \bigwedge_{a+b=c+1} \neg(e \geq a \wedge f \geq b)$$

where the parameters a and b range over $\mathbb{Z}$, satisfying $l(e) \leq a \leq u(e)$ and $l(f) \leq b \leq u(f)$.

Proof. First of all, $(e \leq a \vee f \leq b)$ is equivalent to $\neg(e > a \wedge f > b)$, and to $\neg(e \geq a' \wedge f \geq b')$ as well, with $a' = a + 1$ and $b' = b + 1$. Hence, $l(e) - 1 \leq a \leq u(e) - 1$ and $l(f) - 1 \leq b \leq u(f) - 1$ iff $l(e) \leq a' \leq u(e)$ and $l(f) \leq b' \leq u(f)$. Finally, $a + b = c - 1$ iff $a' + b' = c + 1$. □

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

The following proposition states the correctness of our encoding.

Proposition 2 *For any LIA expression* $\sum_{i=1}^{m} a_i x_i$ *and for any integer* $c \geq l(\sum_{i=1}^{m} a_i x_i)$*, the following holds:*

$$\sum_{i=1}^{m} a_i x_i \leq c \qquad \Longleftrightarrow$$
$$\bigwedge_{\sum_{i=1}^{m} b_i = c+m-1} \neg\Big((a_1 x_1 \geq b_1)^{\#} \wedge \cdots \wedge (a_m x_m \geq b_m)^{\#}\Big)$$

where the parameters b_i *range over* $\mathbb{Z}$ *satisfying* $l(a_i x_i) \leq b_i \leq u(a_i x_i)$*, and the translation* $(\,)^{\#}$ *is defined as follows:*

$$(ax \geq b)^{\#} = \begin{cases} x \geq \left\lceil \dfrac{b}{a} \right\rceil & \textit{if} \quad a > 0 \\ x \leq \left\lfloor \dfrac{b}{a} \right\rfloor & \textit{if} \quad a < 0 \end{cases}$$

Proof. The satisfiability of $\sum_{i=1}^{m} a_i x_i \leq c$ is equivalent to the satisfiability of $\bigwedge_{\sum_{i=1}^{m} b_i = c+m-1} \neg((a_1 x_1 \geq b_1) \wedge \cdots \wedge (a_m x_m \geq b_m))$ from Corollary 1, and the satisfiability of each $a_i x_i \geq b_i$ is equivalent to the satisfiability of $(a_i x_i \geq b_i)^{\#}$. □

3.2 Signed SAT into SAT

Translating signed CNF formulas into satisfiability equivalent Boolean CNF formulas has been proposed in [4, 10]. It amounts to interpreting signed literals as Boolean literals, and adding, for each propositional variable, the following Boolean clauses:[‡]

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

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.

[‡] 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.

4 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). 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 [21] for a survey on this approach.

Among the theories considered in the SMT library [8] we are interested in integer numbers and fixed size bit vectors, and more specifically in the following logics:

- QF_LIA: quantifier-free *linear integer arithmetic*. In essence, closed quantifier-free formulas with Boolean 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 Section 3).

4.1 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 v2.0 under QF_LIA is as follows:*

```
(set-logic QF_LIA)
(declare-fun x11 () Int)
                ...
(declare-fun x33 () Int)
(assert
 (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)))
 )
)
(check-sat)
```

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.

4.2 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: $(\texttt{and } (\geq \texttt{x 1}) \ (\leq \texttt{x } |\texttt{N}|))$.

4.3 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 not only valid for monosigned and regular literals, but for arbitrary signed literals $S_i{:}x$ because 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 formulation is:

```
(set-logic QF_BV)
(declare-fun x1 () Bool)
(declare-fun x2 () Bool)
...
(declare-fun b_x1 () (_ BitVec 8))
(declare-fun b_x2 () (_ BitVec 8))
...
(assert
 (and
  (or x1 ...)
  (or x2 ...)
  ...
  (=>      x1  (= b_x1 #b00000011))
  (=> (not x1) (= b_x1 #b11111100))
  (=>      x2  (= b_x2 #b00000101))
  (=> (not x2) (= b_x2 #b11111010))
  (not (= (bvand b_x1 b_x2) #b00000000))
 )
)
(check-sat)
```

5 SMT-BASED AUTOMATED THEOREM PROVING IN INFINITELY-VALUED LOGICS

The significance of signed CNF formulas is not limited to provide a suitable and effective problem solving framework. Actually, they originated in the area of automated theorem proving in many-valued logics, where it was shown that the satisfiability problem of any finitely-valued logic is polynomially reducible to the satisfiability problem of signed CNF formulas. Unfortunately, signed CNF formulas are not well-suited for dealing with infinitely valued logics too. Therefore, it is not clear how infinitely-valued logics can take advantage from SAT solvers.

The aim of this section is to show that SMT technology can indeed be used to build efficient automated theorem provers in infinitely-valued logics, taking Łukasiewicz infinitely-valued logic as a case study. As we will see, infinitely-valued logics can be encoded into SMT in a natural and compact way. In particular, we are interested in the real numbers theory, namely [8]:

- QF_LRA logic: quantifier-free *linear real arithmetic*. In essence, closed quantifier-free formulas with Boolean combinations of inequalities between linear polynomials over real variables.

On the other hand, the mature state of SMT technology leads to very efficient theorem provers, and it would be very costly to duplicate the same efforts for developing efficient theorem provers based on a tableaux calculus [18] or a sequent calculus [12].

The section is organized as follows. We first formally define Łukasiewicz infinitely-valued logic. Then, we define an SMT-based theorem prover for Łukasiewicz logic. Finally, we report on experimental results that provide empirical evidence of the good performance profile of the defined SMT-based theorem prover.

5.1 Łukasiewicz logic

The syntax and semantics of Łukasiewicz infinitely-valued logic is defined as follows:

Definition 8 *A propositional language is a pair* $\mathbb{L} =< \Theta, \alpha >$*, where* Θ *is a set of logical connectives and* $\alpha : \Theta \rightarrow \mathbb{N}$ *defines the arity of each connective. Connectives with arity 0 are called logical constants. If* $\Theta = \{\theta_1, \ldots, \theta_r\}$*, in the following we will denote the propositional language* $< \Theta, \alpha >$ *with* $< \theta_1/\alpha(\theta_1), \ldots, \theta_r/\alpha(\theta_r) >$*.*

Given a propositional signature Σ, the set L_Σ of $\mathbb{L}$-formulas over Σ is inductively defined as the smallest set with the following properties:

1. $\Sigma \subseteq L_\Sigma$.
2. *If* $\theta \in \Theta$ *and* $\alpha(\theta) = 0$, *then* $\theta \in L_\Sigma$.
3. *If* $\phi_1, \ldots, \phi_m \in L_\Sigma$, $\theta \in \Theta$ *and* $\alpha(\theta) = m$, *then* $\theta(\phi_1, \ldots, \phi_m) \in L_\Sigma$.

Definition 9 *The language of Łukasiewicz logic is given by*

$$\mathbb{L}_{\text{Łuk}} =< \neg/1, \rightarrow/2, \wedge/2, \vee/2, \odot/2, \oplus/2 > .$$

Subsets of the set of connectives such as $\{\rightarrow, \neg\}$ and $\{\oplus, \neg\}$ are functionally complete. Nevertheless, we consider the whole set of connectives to illustrate how they are encoded into SMT. We refer to $\neg$ as negation, refer to $\rightarrow$ as implication, refer to $\wedge$ as weak conjunction, refer to $\vee$ as weak disjunction, refer to $\odot$ as strong conjunction, and refer to $\oplus$ as strong disjunction.

Definition 10 *If* $\mathbb{L} =< \Theta, \alpha >$ *is a propositional language,* N *is a truth value set and* A *assigns to each* $\theta \in \Theta$ *a function* $A_\theta : N^{\alpha(\theta)} \rightarrow N$, *then we call a pair* $\mathbf{A} =< N, A >$ *a matrix for* $\mathbb{L}$.

A pair $\mathcal{L} =< \mathbb{L}, \mathbf{A} >$ *consisting of a propositional language and a matrix is called a many-valued logic.*

Many-valued logics are equipped with a non-empty subset D of N called the designated truth values which are the truth values that are considered to affirm satisfiability.

Definition 11 *The Łukasiewicz infinitely-valued logic is the many-valued logic such that* N *is the real unit interval* $[0, 1]$, $\mathbb{L} = \mathbb{L}_{\text{Łuk}}$, $D = \{1\}$, *and the matrix is given by:*

$$\begin{aligned}
&A_\neg(x) = 1 - x \\
&A_\rightarrow(x, y) = \min\{1, 1 - x + y\} \\
&A_\wedge(x, y) = \min\{x, y\} \\
&A_\vee(x, y) = \max\{x, y\} \\
&A_\odot(x, y) = \max\{0, x + y - 1\} \\
&A_\oplus(x, y) = \min\{1, x + y\}
\end{aligned}$$

Definition 12 *Let* $\mathcal{L}$ *be a many-valued logic. An interpretation is a function* $I : \Sigma \rightarrow N$. I *is extended to arbitrary formulas* ϕ *in the usual way:*

1. *If* ϕ *is a logical constant, the* $I(\phi) = A(\phi)$.

2. *If* $\phi = \theta(\phi_1, \ldots, \phi_r)$ *, then* $I(\theta(\phi_1, \ldots, \phi_r)) = A_\theta(I(\phi_1), \ldots, I(\phi_r))$.

A formula ϕ *of the Łukasiewicz infinitely-valued logic is satisfiable iff there is an interpretation such that* $I(\phi) = 1$*, and it is a tautology iff* $I(\phi) = 1$ *for every interpretation.*

5.2 The SMT-based Theorem Prover

Our aim is to develop a theorem prover capable of determining whether a formula ϕ of the Łukasiewicz infinitely-valued logic is a tautology. To this end, we develop a satisfiability checker for Łukasiewicz logic, and we ask whether there is an interpretation I such that $I(\phi) < 1$. If such an interpretation does not exist, then ϕ is a tautology.

Figure 2 shows the code of the SMT-based theorem prover for the Łukasiewicz logic in the SMT-LIB language v2.0 under QF_LRA. We can find three sections. The first one determines the logic to be used: quantified free linear real arithmetic (QF_LRA). The second one declares the connectives of Łukasiewicz logic, and the variables occurring in the formula. Notice that the variable domains are restricted to the real unit interval. After the definition of the `min` and `max` functions, all the connectives except negation use the `ite` (if-then-else) SMT function: `(ite Bool s s)` returns its second argument or its third depending on whether its first argument is true or not.

The third one contains the following query: Is there an interpretation I of the formula $\phi = (x \oplus x) \vee (y \oplus y) \rightarrow (x \vee y) \oplus (x \vee y)$ such that $I(\phi) < 1$. ϕ is a tautology because the theorem prover returns unsatisfiable. If we would like to determine whether another formula is a tautology, we should only replace the formula and leave the rest of the program unchanged.

5.3 Experimental Results

We performed an empirical comparison between our theorem prover and the three theorem provers for the Łukasiewicz logic used in [20], which correspond to implementations of the theorem provers of Hähnle [16] based on mixed integer programming, Olivetti [18], based on tableaux, and Ciabattoni et al. [12], based on a hypersequent of relations calculus.

We solved the benchmarks used in [20], which are defined as follows:

- Formula 1: $A^n \vee B^n \rightarrow (A \vee B)^n$
- Formula 2: $(A \vee B)^n \rightarrow A^n \vee B^n$

where $A^1 = A$ and $A^n = A \odot A^{n-1}$.

```
(set-logic QF_LRA)

; min(x,y)
(define-fun min ((x Real) (y Real)) Real
 (ite (> x y) y x))
; max(x,y)
(define-fun max ((x Real) (y Real)) Real
 (ite (> x y) x y))
; strong disjunction: sdis(x,y) = min{1,x+y}
(define-fun sdis ((x Real) (y Real)) Real
 (min 1 (+ x y)))
; strong conjuction: scon(x,y) = max{0,x+y-1}
(define-fun scon ((x Real) (y Real)) Real
 (max 0 (- (+ x y) 1)))
; weak disjunction: wdis(x,y) = max{x,y}
(define-fun wdis ((x Real) (y Real)) Real
 (max x y))
; weak conjunction: wcon(x,y) = min{x,y}
(define-fun wcon ((x Real) (y Real)) Real
 (min y x))
; negation: neg(x) = 1 - x
(define-fun neg ((x Real)) Real
 (- 1 x))
; implication: impl(x,y) = min{1,1-x+y}
(define-fun impl ((x Real) (y Real)) Real
 (min 1 (- (+ 1 y) x)))
(declare-fun x () Real)
(declare-fun y () Real)
(assert (>= x 0))
(assert (<= x 1))
(assert (>= y 0))
(assert (<= y 1))

(assert (< (impl (wdis (scon x x) (scon y y))
                 (scon (wdis x y) (wdis x y))) 1))
(check-sat)
```

FIGURE 2
Code of the SMT-based theorem prover for the Łukasiewicz infinitely-valued logic.

The experiments range from $n = 2$ to $n = 12$. We set a cutoff time of 1 hour on a machine with 2.5 GHz and 0.5G mem. We ran the theorem provers used in [20] on SWI-Prolog [24] (version 5.10.5) using the `clpqr` (Constraint Logic Programming over Rationals and Reals) library, which is a port of the CLP(Q,R) system of Sicstus Prolog. The SMT solver we used in our approach is Z3 [13] (version 3.2) with the QF_LRA logic.

The results obtained are shown in Table 1. Run time is in seconds. Failed tests are indicated with *. As pointed out in [20], this may be due to problems with the `clpq` constraint library.

As we can see, our approach with SMT clearly outperforms the rest of the approaches. This result validates our thesis that SMT technology allows to build efficient automated theorem provers for infinitely-valued logics, such as Łukasiewicz logic. In order to check that there is a value of n for which the formula becomes harder in our approach we have added an extra line with $n = 500$.

SMT solvers are designed to efficiently deal with problems that have a Boolean structure over atoms of some theories such as the real numbers. Notice that the `ite` functions build the Boolean structure on the predicates on the theory of reals. SMT solvers typically integrate a SAT solver, which efficiently deals with the Boolean structure of the formula, and (one or more) theory solvers which efficiently deal with the theory predicates.

6 CONCLUSIONS

Our work proposes the use of SAT and SMT technology to deal with many-valued logics. We have first focused on finitely-valued logics, and have defined a new many-valued modeling language that extends signed CNF formulas with LIA constraints. The resulting language (MVL-formulas) allows to model a wide range of practical combinatorial problems in a natural and compact way.

Then, 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 either 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 since 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

	Hähnle		Olivetti		Ciabonetti et al.		SMT	
n	(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
2	0.49	0.42	0.01	0.02	0.34	0.01	0.02	0.03
3	1.1	1.5	0.06	0.20	0.04	0.03	0.03	0.03
4	3	5	0.17	1.3	0.17	0.10	0.03	0.03
5	8	15	0.45	7	1.2	0.22	0.03	0.03
6	24	52	1.1	31	7	0.61	0.03	0.03
7	81	195	2.7	125	*	1.6	0.03	0.03
8	281	3578	6	462	*	4	0.03	0.03
9	2409	$>$ 1h	15	1628	*	10	0.04	0.04
10	$>$ 1h	$>$ 1h	33	$>$ 1h	*	26	0.04	0.04
11	$>$ 1h	$>$ 1h	77	$>$ 1h	*	63	0.04	0.04
12	$>$ 1h	$>$ 1h	181	$>$ 1h	*	149	0.04	0.04
500	$>$ 1h	$>$ 1h	$>$ 1h	$>$ 1h	*	$>$ 1h	436	288

TABLE 1
Comparison of theorem provers for Łukasiewicz logic. Run time in seconds.

translation to both SAT and SMT. We plan to provide direct translations into CP solver languages as well.

It is worth noticing that, for the first time, we have 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.

Besides, we have established, for the first time, a link between signed CNF formulas 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 many-valued SAT problems. The next step is to design and conduct an empirical evaluation of these encodings.

We have also focused on infinitely-valued logics. In particular, we have shown that it is possible to easily build efficient automated theorem provers

based on the SMT technology. As a case study we have considered the Łukasiewicz infinitely-valued logic, but our approach can be extended to other infinitely-valued logics. Actually, after the acceptance of the present paper, SMT technology has been used to build automated theorem provers for other relevant infinitely-valued logics [2, 23]. Moreover, the reported experimental investigation shows that the proposed SMT approach clearly outperforms other existing theorem provers for Łukasiewicz infinitely-valued logic.

7 ACKNOWLEDGEMENTS

Research partially supported by the Generalitat de Catalunya under grant AGAUR 2009-SGR-1434, the *Ministerio de Economía y Competividad* research projects AT CONSOLIDER CSD2007-0022, INGENIO 2010, TIN2008-04547, TIN2009-14704-C03-01, TIN2010-20967-C04-01/03, and Newmatica INNPACTO IPT-2011-1496-310000 (funded by MICINN until 2011).

REFERENCES

[1] Carlos Ansótegui, Miquel Bofill, Felip Manyà, and Mateu Villaret. (2011). Extending multiple-valued clausal forms with linear integer arithmetic. In *Proceedings, 42nd International Symposium on Multiple-Valued Logics (ISMVL), Tuusula, Finland*, pages 230–235. IEEE CS Press.

[2] Carlos Ansótegui, Miquel Bofill, Felip Manyà, and Mateu Villaret. (2012). Building automated theorem provers for infinitely-valued logics with satisfiability modulo theory solvers. In *Proceedings, 42nd International Symposium on Multiple-Valued Logics (ISMVL), Victoria, Canada*, pages 25–30. IEEE CS Press.

[3] Carlos Ansótegui, Jose Larrubia, Chu Min Li, and Felip Manyà. (2007). Exploiting multivalued knowledge in variable selection heuristics for SAT solvers. *Annals of Mathematics and Artificial Intelligence*, 49(1-4):191–205.

[4] Carlos Ansótegui and Felip Manyà. (2004). 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.

[5] Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. (2010). New insights into encodings from MaxCSP into partial MaxSAT. In *Proceedings, 40th International Symposium on Multiple-Valued Logics (ISMVL), Barcelona, Spain*, pages 46–52. IEEE CS Press.

[6] Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. (2012). Efficient encodings from CSP into SAT, and from MaxCSP into MaxSAT. *Multiple-Valued Logic and Soft Computing*, 19(1-3):3–23.

[7] Matthias Baaz and Christian G. Fermüller. (1995). Resolution-based theorem proving for many-valued logics. *Journal of Symbolic Computation*, 19:353–391.

[8] Clark Barrett, Aaron Stump, and Cesare Tinelli, (2010). The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`.

[9] Bernhard Beckert, Reiner Hähnle, and Felip Manyà. (2000). The SAT problem of signed CNF formulas. In David Basin, Marcello D'Agostino, Dov Gabbay, Seán Matthews, and Luca Viganò, editors, *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 61–82. Kluwer, Dordrecht.

[10] Ramón Béjar, Reiner Hähnle, and Felip Manyà. (2001). 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.

[11] Ramón Béjar, Felip Manyà, Alba Cabiscol, Cèsar Fernández, and Carla Gomes. (2007). Regular-SAT: A many-valued approach to solving combinatorial problems. *Discrete Applied Mathematics*, 155(12):1613–1626.

[12] Agata Ciabattoni, Christian G. Fermüller, and George Metcalfe. (2005). Uniform rules and dialogue games for fuzzy logics. In Franz Baader and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 11th International Conference, LPAR 2004, Montevideo, Uruguay, March 14-18, 2005, Proceedings*, volume 3452 of *LNCS*, pages 496–510. Springer.

[13] Leonardo Mendonça de Moura and Nikolaj Bjørner. (2008). Z3: An efficient SMT solver. In *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340.

[14] Reiner Hähnle. (1991). 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.

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

[16] Reiner Hähnle. (1994). Many-valued logic and mixed integer programming. *Annals of Mathematics and Artificial Intelligence*, 12:231–263.

[17] Siddhartha Jain, Eoin O'Mahony, and Meinolf Sellmann. (2010). 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.

[18] Nicola Olivetti. (2003). Tableaux for Łukasiewicz infinitely-valued logic. *Studia Logica*, 73(1):81–111.

[19] Steven David Prestwich. (2009). CNF encodings. In Armin Biere, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, pages 75–97. IOS Press.

[20] Robert Rothenberg. (2007). A class of theorems in Łukasiewicz logic for benchmarking automated theorem provers. In *16th International Conference, TABLEAUX 2007, Aix en Provence,France*. Position paper.

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

[22] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara. (2009). Compiling finite linear CSP into SAT. *Constraints*, 14(4):254–272.

[23] Amanda Vidal, Félix Bou, and Lluis Godo. (2012). An SMT-based solver for continuous t-norm based logics. In *Proceedings of the 6th International Conference on Scalable Uncertainty Management (SUM), Marburg, Germany*, pages 633–640. Springer LNAI 7520.

[24] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. (2010). Swi-prolog. *CoRR*, abs/1011.5332.