Efficient Deduction in Many-Valued Logics

Reiner Hähnle
University of Karlsruhe
Institute for Logic, Complexity and Deduction Systems
76128 Karlsruhe, Germany, reiner@ira.uka.de

Abstract

This paper tries to identify the basic problems encountered in automated theorem proving in many-valued logics and demonstrates to which extent they can be currently solved. To this end a number of recently developed techniques are reviewed. We list the avenues of research in many-valued theorem proving that are in our eyes the most promising.

1 Introduction

The purpose of this note is to review a number of techniques that lead to a computationally adequate representation of the search space of many-valued logics and to identify the avenues of research in many-valued theorem proving that are in our eyes the most promising. We do not mention the large number of possible applications of many-valued theorem proving, but refer to [15] for an extensive list of applications and to [18] for a case study.

If one is doing many-valued deduction, typically a number of problems that are not as much prominent in classical deduction have to be addressed:

1. The number of case distinctions is much larger due to the increased number of truth values.

2. The amount of redundancy in deductions is much bigger. Typically, many-valued connectives show a certain degree of regularity and one has to find ways of how to exploit this.

3. In general, internal normal forms (that is, normal forms based solely on connectives from the logic under consideration) are not available.

4. In the case of infinitely-valued logics there is the problem to find a finite representation of the search space.

Throughout this paper we assume that the reader is familiar with the basic notions of computational logic and with the semantic tableau proof procedure in particular. A good reference for the required background is [10]. We will use a standard syntax for propositional and first-order logic, here and there enriched with some new unary and binary operator symbols. We use $p, q, r, p_1, q_1, r_1, \ldots$ for propositional variables and predicate names; $c, c_i, t, t_j, \ldots$ for constants and terms; $\phi, \psi, \ldots$ for propositional and first-order formulas; F for unspecified connectives.

2 Many-Valued Logic

Definition 1 (Syntax) $\mathbf{L}$ *is a propositional language with propositional variables* $\mathbf{L}_0$ *and finitary connectives* $\mathbf{F}$. ■

Definition 2 (Truth Values, Semantic, Logic) *Let N be the set of* **truth values***; for definiteness in the finitely-valued case, take equidistant rational numbers, i.e.* $N = \left\{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\right\}$ *and set* $n = |N|$*; in the infinitely-valued case let* $N = \mathbb{Q} \cap [0,1]$ *and* $n = \infty$*. Connectives* $F \in \mathbf{F}$ *are interpreted as functions with range and domain over N, in other words, if k is the arity of F we associate a function* $f : N^k \to N$ *with F which we call the* **interpretation** *of F. Let* $\mathbf{f}$ *be the family of functions over N associated with connectives in* $\mathbf{F}$*. Then we call* $\mathbf{A} = \langle N, \mathbf{f} \rangle$ *an n-valued matrix for* $\mathbf{L}$ *and* $\mathcal{L} = \langle \mathbf{L}, \mathbf{A} \rangle$ *an* **n-valued propositional logic***. ■

Definition 3 (Valuation) *Let $\mathcal{L}$ be an n-valued propositional logic. A* **valuation** *for $\mathcal{L}$ is a function* $v : \mathbf{L}_0 \to N$*. As usual, v can be uniquely extended to a homomorphism from* $\mathbf{L}$ *to N via*

$$v(F(\phi_1, \ldots, \phi_k)) = f(v(\phi_1), \ldots, v(\phi_k))$$

where f is the interpretation of F. ■

Definition 4 (S-Satisfiability, -Consequence) *If $S \subseteq N$, and $\mathcal{L}$ is an n-valued propositional logic then call a formula $\phi \in \mathbf{L}$* S-**satisfiable** *iff there is a valuation v such that $v(\phi) \in S$. Call ϕ an* S-**tautology** *iff $v(\phi) \in S$ for all valuations. $\psi \in \mathbf{L}$ is an S-consequence of $\Phi \subseteq \mathbf{L}$, denoted by $\Phi \models_S \psi$, if every valuation that S-satisfies Φ also S-satisfies ψ.* ■

We can extend our presentation to quantified logic in the following way:

Definition 5 (First-Order Syntax) $\mathbf{L}^1$ *is a first-order language which is constructed from a propositional language $\mathbf{L}$ in the usual way by replacing the set of propositional variables with atoms of the form $p(t_1, \ldots, t_m)$, the t_i being from a term language $\mathbf{T}$ made up of sets E_i that contain function symbols of arity i for $i \geq 0$ and object variables V; $p \in P$, where $P = \bigcup_{i \geq 0} P_i$ is a non-empty set of predicate symbols and each P_i contains i-ary predicates. Moreover, we allow quantified formulas: if $x \in V$, $\phi \in \mathbf{L}^1$ and $Q \in \{Q_1, \ldots, Q_r\}$ then $(Qx)\phi \in \mathbf{L}^1$.* ■

Definition 6 (First-Order Semantics/Logic) *A* **domain** *D, a* **variable assignment** *β, an* **interpretation** *I, a* **model** *$\mathbf{M} = \langle D, I \rangle$, a* **first-order valuation** *$v_{M,\beta}$,* **first-order satisfiability**, *and* **validity** *are defined as in classical logic (just note that predicates are evaluated in N, hence $I(p) : D^i \to N$). The semantics of quantifiers is given via their* **distribution** *$d_Q : 2^N \to N$ of truth values as the quantified variable runs through D:*

$$v_{M,\beta}((Qx)\phi) = d_Q(\{v_{M,\beta^u_x}(\phi) \mid u \in D\})$$

Given $\mathbf{d} = \{d_{Q_1}, \ldots, d_{Q_r}\}$ and an n-valued propositional logic $\mathcal{L}$, an n-**valued first-order logic** *$\mathcal{L}^1$ associated with $\mathcal{L}$ is defined by the triple $\langle \mathbf{L}^1, \mathbf{A}, \mathbf{d} \rangle$* ■

Example 7 Consider the propositional logic with a unary negation connective $\neg$, binary connectives $\oplus, \sqcup, \sqcap$ called truncated sum, disjunction and conjunction, respectively. For arbitrary N, the semantics is given by $\neg i := 1 - i$, $i \oplus j := \min\{1, i + j\}$, $i \sqcup j := \max\{i, j\}$, $i \sqcap j := \min\{i, j\}$.

In the finitely-valued case we can specify a first-order logic based on these connectives and $\forall$ and $\exists$ by stipulating $d_\exists := \max$ and $d_\forall := \min$ (where max, min are interpreted naturally). We call the family of logics just defined **Łukasiewicz Logics**. ■

The general problem in many-valued deduction can now be formulated as follows: given a collection of sets of truth values $S_1, \ldots, S_k \subseteq N$, a many-valued (first-order) logic $\mathcal{L}$ and closed formulas $\phi_1, \ldots, \phi_k \in \mathbf{L}$, is there a model $\mathbf{M}$ which simultaneously S_i-satisfies ϕ_i for all $1 \leq i \leq k$? Hence, we assume that there is some kind of deduction theorem which gives us a translation $\mathbf{Tr} : 2^L \times \mathbf{L} \to 2^{L^*}$ from S-consequence to (simultaneous) S-satisfiability such that $\Phi \models_S \psi$ iff $\mathbf{Tr}(\Phi, \psi)$ is satisfiable. Such deduction theorems indeed exist for many logics. See [8] for some non-trivial examples and further references.

3 Sets as Signs

If one is seeking for efficient many-valued deduction a simple, but very useful device is needed: analogously to the signs $\mathbf{T}$ and $\mathbf{F}$ in classical semantic tableaux [27] one introduces subsets of the set of truth values as a meta-logical notation in order to denote restrictions on the truth value a formula may take on.

Definition 8 (Signed Formula) *Let $S \subseteq N$ and $\phi \in \mathbf{L}$. Then we call the expression $S\ \phi$ a* **signed formula** *and we denote the set of all signed formulas with $\mathbf{L}^*$. $S\ \phi$ is satisfiable iff ϕ is S-satisfiable.* ■

A semantic tableau-based proof procedure for finitely-valued logics based on truth value sets as signs was first introduced in [12].

Let us look at the example in Table 1 in order to see what the sets-as-signs approach (as we prefer to call it) can gain.

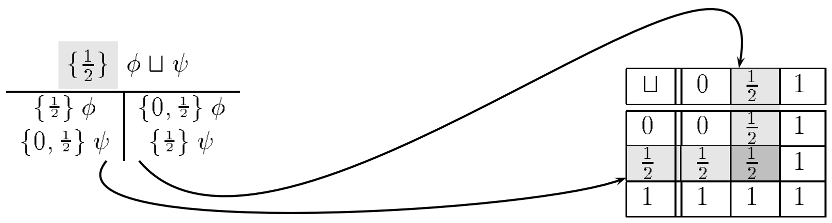
Semantic tableau rules correspond to a *classical* DNF representation of the premise. Each rule extension is a conjunction of signed subformulas and represents a partial covering of those truth table entries that occur in the sign of the formula in the premise (in the example indicated by the arcs). The union of all partial coverings (that is the collection of all rule extensions) characterizes exactly those entries.

Obviously, using sets-as-signs (in contrast to single truth values) can shorten the rules considerably. The rule from Table 1, but with singleton signs only, becomes

$$\begin{array}{c|c|c} \multicolumn{3}{c}{\frac{1}{2}\ \phi \sqcup \psi} \\ \hline \frac{1}{2}\ \phi & \frac{1}{2}\ \phi & 0\ \phi \\ 0\ \psi & \frac{1}{2}\ \psi & \frac{1}{2}\ \psi \end{array}$$

It is clear that nested application of such rules can result in exponential differences between sets-as-signs and singleton signs.

Table 1: Sets-as-signs rule and truth table for $\sqcup$ and $n = 3$.



$$\begin{array}{c|c} \multicolumn{2}{c}{\{\frac{1}{2}\}\ \phi \sqcup \psi} \\ \hline \{\frac{1}{2}\}\ \phi & \{0, \frac{1}{2}\}\ \phi \\ \{0, \frac{1}{2}\}\ \psi & \{\frac{1}{2}\}\ \psi \end{array}$$

$\sqcup$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

Although the effect of using sets-as-signs is dramatic also in practice (cf. Section 7) the idea was not systematically exploited before [12].[1]

Note that a tableau for a many-valued formula employing sets-as-signs is still a classical tableau on the meta level. Hence, truth value sets as signs are just meta connectives which are suitably chosen in order to allow for an efficient representation of many-valued models.[2] Therefore, the sets-as-signs approach is useful not only in the context of tableau-based theorem proving, from where it evolved, but also, as we shall see in Sections 5 and 6, within the scope of other theorem proving paradigms. See [15] for more examples.

In the presence of quantifiers the usefulness of sets-as-signs becomes even more striking. Let us first give a computational description of the distribution of a quantifier Q: If $v((Qx)\phi(x)) = i$ holds due to the fact that $d_Q(\{i_1, \ldots, i_k\}) = i$ for a certain distribution of truth values, where $\{i_1, \ldots, i_k\} \subseteq N$ and $i \in N$, this means that

1. $v(\phi(t_1)) = i_1, \ldots, v(\phi(t_k)) = i_k$ must hold for certain terms $t_1, \ldots, t_k$, and

2. $v(\phi(t)) \in \{i_1, \ldots, i_k\}$ must hold for any term t.

These conditions can be conveniently expressed in rule format with Skolem terms and signs:

$$\begin{array}{c|c|c} \multicolumn{3}{c}{I\ (Qx)\phi(x)} \\ \hline \{i_{11}\}\ \phi(c_1) & & \{i_{m1}\}\ \phi(c_1) \\ \vdots & \cdots & \vdots \\ \{i_{1k_1}\}\ \phi(c_{k_1}) & & \{i_{mk_m}\}\ \phi(c_{k_m}) \\ I_1\ \phi(t_1) & & I_m\ \phi(t_m) \end{array}$$

[1][29, 23] are the only approaches employing the idea at all that we are aware of. Both, however, are restricted to specific logics. For a complete historical account and bibliography on many-valued theorem proving, see [15].

[2]In non-classical theorem proving the existence and choice of a suitable meta language is crucial, cf. [9, 25].

Here $d_Q^{-1}(I) = \{I_1, \ldots, I_m\}$, $I_j = \{i_{j1}, \ldots, i_{jk_j}\}$, the $c_1, c_2, \ldots$ are new Skolem constants[3], and the $t_1, \ldots, t_m$ are arbitrary terms.

For each I_j such that $d_Q(I_j) \in I$ there is a rule extension wherein Condition 1 above is expressed by the first k_j formulas and Condition 2 by the last formula. We note that in the case of $I_j = \{i_j\}$ for some j for the corresponding extension it is sufficient to contain the single signed formula $I_j\ \phi(t)$, and one can always delete tautological signed formulas of the form $N\,\phi(t)$. We stress that condition 2 above is extremely complicated to formulate as a rule when only singleton signs are available, cf. [7]. Again, the use of sets-as-signs can lead to exponential speed-ups. On the other hand, in the formulation above we have still up to $2^n - 2$ extensions in a rule since for each set of truth values in $d_Q^{-1}(I)$ exactly one extension must be generated.

If we compute, for example, the tableau rule for $\{0, \frac{1}{2}\}(\forall x)\phi(x)$ in three-valued logic we obtain the rule shown in Table 2.

This rule is obviously not the simplest possible one. In order to obtain it, we encoded each truth value set in $d_\forall^{-1}(\{0, \frac{1}{2}\})$ with Skolem conditions. If we turn this process around, and ask ourselves which distributions of truth values can be encoded using conjunctions of Skolem conditions of the form $I\,\phi(c)$ or $J\,\phi(t)$, where $I, J \subseteq N$, we see that this "Skolem language" is quite powerful. We may encode, for instance, the family of truth value sets defined through the expression $2^N - \{X \mid X \subseteq \{i + \frac{1}{n-1}, \ldots, 1\}\}$ by $\{0, \ldots, i\}\,\phi(c)$ for each $i \in N$. Hence, the rule shown in Table 2 can be simpified to

$$\frac{\{0, \frac{1}{2}\}\ (\forall x)\phi(x)}{\{0, \frac{1}{2}\}\ \phi(c)} \qquad (*)$$

As has already been pointed out by Carnielli [6, p. 488], even for singleton signs it is as yet an unsolved problem to find minimal rules for distribution quantifiers automatically in a feasible way that is without

[3]It is sufficient for Skolem constants to be new only wrt to the current branch [27].

Table 2: Tableau rule for $\{0, \frac{1}{2}\}\ (\forall x)\phi(x)$ in three-valued logic.

$\{0, \frac{1}{2}\}\ (\forall x)\phi(x)$					
	$\{0\}\ \phi(c)$	$\{0\}\ \phi(c)$	$\{0\}\ \phi(c)$		$\{\frac{1}{2}\}\ \phi(c)$
	$\{\frac{1}{2}\}\ \phi(d)$	$\{\frac{1}{2}\}\ \phi(d)$	$\{1\}\ \phi(d)$		$\{1\}\ \phi(d)$
		$\{1\}\ \phi(e)$			
$\{0\}\ \phi(t_1)$	$\{0, \frac{1}{2}\}\ \phi(t_2)$		$\{0, 1\}\ \phi(t_4)$	$\{\frac{1}{2}\}\ \phi(t_5)$	$\{\frac{1}{2}, 1\}\ \phi(t_6)$

enumerating all possible rules. What would be needed is a sound and complete set of rewrite rules over the "Skolem language" defined above. In the next section, however, we develop a notion which, at least in the case of the standard quantifiers $\forall$ and $\exists$, leads to a satisfactory solution.

4 Regular Logics

It turns out that a number of many-valued logics have particularly simple computational properties. Working with sets-as-signs is again useful for identifying them. Let us start with the observation that if we omit the $\oplus$ connective in the logic defined in Section 2 and consider only the following signs

$$\boxed{\geq j} \quad := \quad [j, 1] \cap N \qquad\qquad \boxed{\leq j} \quad := \quad [0, j] \cap N$$

then all tableau rules have either the shape of α rules or of β rules in the sense of Smullyan [27]. Moreover, the signs occurring in the conclusion of the rules are again of the form $\boxed{\geq j}$, $\boxed{\leq j}$. Let us call these signs **regular signs**.

The main reasons for this very simple shape are (i) the restricted form of the signs; (ii) the truth values corresponding to the truth table entries are monotonically increasing or decreasing, starting from one corner. We express these constraints in a formal definition:

Definition 9 (Circle, Corner [13]) *A metric d on N^k can be defined by*

$$d(\vec{x}, \vec{y}) = \max_{1 \leq i \leq k} |x_i - y_i|$$

for $\vec{x}, \vec{y} \in N^k$. For $r \in N$ we define the **circle** *in N^k with center $\vec{x}$ and radius r as the set*

$$c_{\vec{x},r} = \{\vec{y} |\ \vec{y} \in N^k \text{ and } d(\vec{x}, \vec{y}) = r\}.$$

We define the **corner set** *of N^k as*

$$I^k = \{\vec{x} |\ x_i \in \{0, 1\}, 1 \leq i \leq k\} = \{0, 1\}^k.$$

■

Definition 10 (Regular Logic [13]) *A k-ary connective f is called* **regular** *iff there is an $\vec{x} \in I^k$ such that*

1. *for all $r \in N$ the set $\{f(\vec{y}) | \vec{y} \in c_{\vec{x},r}\}$ is a singleton, say $\{x_r\}$, and*
2. *the sequence $x_0, \ldots, x_1$ composed of these x_r is monotonic (either increasing or decreasing) wrt the natural order on N.*

We call $\vec{x}$ the **starting point** *of f.*

A many-valued logic with only regular connectives, standard quantifiers, and queries restricted to regular signs is called **regular logic**. ■

The logic defined in Section 2 without the $\oplus$ connective is regular. In regular logics all tableau rules for propositional connectives are α rules or β rules, all quantifier rules are γ or δ rules in the sense of Smullyan [27], see below. Moreover, these rules can be computed straightforwardly in a schematic manner from the semantics, see [13, 15] for details.

The attractive feature of regular logics is that they can be handled almost like classical logic, but they still are quite expressive.[4] For instance, the connective $\oplus$ defined above is not regular, however it can be easily composed of regular connectives. To see this, we first define for $n = 3$ the connective $\curlywedge$ via the truth table on the left. On the right the truth table of the target function $\oplus$:

$\curlywedge$	0	$\frac{1}{2}$	1
0	0	0	0
$\frac{1}{2}$	0	1	1
1	0	1	1

$\oplus$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	1	1
1	1	1	1

It is easy to see that $(\phi \oplus \psi) = ((\phi \sqcup \psi) \sqcup (\phi \curlywedge \psi))$ holds for $n = 3$.[5]

[4] In [15] it is shown that for each n there is a functionally complete regular n-valued logic.

[5] Note, however, that the definition of $\oplus$ becomes increasingly more complex while the number of truth values is growing. Also the size of a formula can increase exponentially during the translation into a regular formula. A remedy to this are the techniques introduced in Section 6.

In the presence of regular signs the standard quantifier rules become also extremely simple (cf. the classical rules in [27]):

$$\frac{\gamma(x)}{\gamma(t)} \qquad \frac{\delta(x)}{\delta(c)}$$

where t is an arbitrary term, c a new Skolem constant, $\boxed{\geq i}\,(\forall x)\phi$ and $\boxed{\leq i}\,(\exists x)\phi$ are formulas of type γ, and $\boxed{\geq i}\,(\exists x)\phi$ and $\boxed{\leq i}\,(\forall x)\phi$ are formulas of type δ.

A full soundness and completeness proof of these rules, which generalize (*), is given in [15].

It turns out that the notion of a regular sign can be naturally extended to arbitrary partially ordered sets and they coincide with the well-known notions of *upset* and *downset* from lattice theory.

Definition 11 (Partially Ordered Set) *A* **partially ordered set** *(briefly,* **poset***) is a pair $\langle P, \preccurlyeq\rangle$, where P is a nonempty set, $\preccurlyeq$ a binary relation on P and (P1)–(P3) below hold:*

(P1) *For any $x \in P$: $x \preccurlyeq x$.*

(P2) *For any $x, y, z \in P$: If $x \preccurlyeq y$ and $y \preccurlyeq z$ then $x \preccurlyeq z$.*

(P3) *For any $x, y \in P$: If $x \preccurlyeq y$ and $y \preccurlyeq x$ then $x = y$.* ∎

Definition 12 (Up-, Downset) *Let $\langle P, \preccurlyeq\rangle$ be a poset. We define for $a \in P$: $\uparrow a := \{x | \; a \preccurlyeq x, x \in P\}$, $\downarrow a := \{x | \; x \preccurlyeq a, x \in P\}$.* ∎

For example, if $\preccurlyeq$ is the natural order on N, we have $\boxed{\geq i} = \uparrow i$, $\boxed{\leq i} = \downarrow i$, $\{i\} = \uparrow i \cap \downarrow i$. Natural candidates for a weaker structure to try out would be finite (complemented and/or distributive) lattices.

Here close connections to work done in theorem proving in paraconsistent annotated logics show up [23].

5 Normal Forms

So far we have mainly talked about short representations of semantic tableau rules for many-valued logic. We have, however, promised that the presented techniques are universally applicable. In order to see this we emphasize the normal form aspect of tableau rules. It has been already remarked that a tableau rule corresponds to a DNF representation of the signed formula in the premise. As an example, let us rewrite the conclusion of the rule for $\{\frac{1}{2}\}\,(\phi \sqcup \psi)$ from Table 1 as $(\{\frac{1}{2}\}\,\phi \wedge \{0, \frac{1}{2}\}\,\psi) \vee (\{0, \frac{1}{2}\}\,\phi \wedge \{\frac{1}{2}\}\,\psi)$. Note that $\wedge, \vee$ are **classical** connectives and the literals $S\phi$ in this DNF are interpreted classically in the obvious way: $S\,\phi$ is classically satisfiable iff ϕ is S-satisfiable.

Hence recursive application of tableau rules to subformulas transforms any finitely-valued signed formula into a **classical** DNF representation based on signed atoms as literals.[6] Such a DNF clause is a conjunction of signed atoms and its satisfiability can be easily checked: $C = S_1\,p_1 \wedge \cdots \wedge S_m\,p_m$ is unsatisfiable iff there are $S_{i_1}\,p_{i_1}, \ldots, S_{i_k}\,p_{i_k}$ such that $p_{i_1} = \cdots = p_{i_k}$ and $\bigcap_{1\leq j\leq k} S_{i_j} = \emptyset$.

With the two basic ingredients (i) DNF transformation and (ii) satisfiability checking of conjunctive paths (which is just another name for a DNF clause) we can apply the techniques of the previous sections to a lot of well-known proof procedures which rest on these properties, for instance, to the connection method, model elimination, model generation, path dissolution, decision diagrams. See [15] for some worked out examples.

On the other hand, the seemingly most successful theorem provers for classical first-order logic are working with CNF-based resolution. In order to achieve a signed CNF instead of DNF, all we have to do is to provide tableau rules that relate to CNF instead of DNF and use free variable versions [10, 3] of the rules for $\forall$ and $\exists$ of the previous section (for other quantifiers the free variable problem is more complex). For the example from the beginning of this section we have the picture summarized in Figure 1. Each rule extension corresponds to a region of the truth table that covers all fields with entries that occur in the premise. The *intersection* of all such coverings (the darkly shaded fields in the table on the righthand side in Figure 1) comprise exactly those fields. The double vertical bars in the rule indicate that it is a CNF rule.

Several optimizations of this approach to CNF transformation are possible. For instance, one can design a structure preserving algorithm extending Tseitin's work [30] for classical logic. The result of such an algorithm is always polynomially bounded wrt to the length of the input [16] in the case of finitely-valued first-order logics with standard quantifiers.

Finally, a number of sound and complete resolution rules can be defined on signed clauses. We give one possible formulation and refer the reader to [16, 25, 24, 1] for further details.

Resolution, parallel version:

[6] In the first-order case we do not know the required number of applications of quantifier rules beforehand, so we can speak only of a DNF approximation.

$$\frac{\{\frac{1}{2}\}\ \phi \sqcup \psi}{\{0,\frac{1}{2}\}\ \psi \ \Big\| \ \begin{array}{c}\{\frac{1}{2}\}\ \phi \\ \{\frac{1}{2}\}\ \psi\end{array} \ \Big\| \ \{0,\frac{1}{2}\}\ \phi}$$

$$\{\tfrac{1}{2}\}\ \phi \sqcup \psi = \{0,\tfrac{1}{2}\}\ \psi \wedge (\{\tfrac{1}{2}\}\ \phi \vee \{\tfrac{1}{2}\}\ \psi) \wedge \{0,\tfrac{1}{2}\}\ \phi$$

$\sqcup$	0	$\frac{1}{2}$	1
0	0	$\frac{1}{2}$	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1
1	1	1	1

Figure 1: Illustration of CNF sets-as-signs rule for $\sqcup$ and $n = 3$.

$$\frac{S_1\, p_1 \vee D_1 \ \cdots \ S_m\, p_m \vee D_m}{(D_1 \vee \cdots \vee D_m)\sigma} \qquad \begin{array}{r}\bigcap_{i=1}^m S_i = \emptyset \\ \sigma \text{ mgu of } p_1, \ldots, p_m\end{array}$$

Merging:

$$\frac{S_1\, p_1 \vee \cdots \vee S_m\, p_m \vee D}{((S_1 \cup \cdots \cup S_m)\, p_1 \vee D)\sigma} \qquad \sigma \text{ mgu of } p_1, \ldots, p_m$$

If we deal with regular logics, only regular signs occur in the literals. In this case a binary resolution rule is sufficient.

An important feature of signed resolution is that many resolution refinements known from classical logic can be either directly applied or can be extended in a suitable way (this has been done in [1] for singleton signs). Examples are deletion of tautologies and pure clauses, UR-resolution, lock resolution, ordered resolution; as a concrete example we state subsumption.

Definition 13 (Signed Subsumption) *Let D, E be signed clauses. D* **is subsumed** *by E iff there is a substitution σ such that for each literal $S_1\, p_1$ in E there is a literal $S_2\, p_2$ in D such that $S_1 \subseteq S_2$ and $p_1\sigma = p_2$.* ∎

6 Integer Programming

Let us restrict our attention to classical propositional logic for the moment. The correspondence between classical propositional formulas in CNF and $0-1$ integer programs is well known and has led to some research on the relationship between logic satisfiability checking and linear optimization [20]. It turns out that a semantic tableau based view leads to a generalization of this relationship. The key idea is to use regular signs and leave the variables in the signs uninstantiated. Hence, we allow rules as the following:

$$\frac{\boxed{\geq i}\ F(\phi_1, \phi_2)}{\begin{array}{c}\boxed{\geq i_1}\ F(\phi_1, \phi_2) \\ \boxed{\geq i_2}\ F(\phi_1, \phi_2)\end{array}}$$

For most instances of i, i_1, i_2 such a rule does not reflect the semantics of F, hence we must impose some additional constraints in order to make it sound. It turns out that constraints in the form of *linear inequalities over the variables occurring in the signs* are sufficient to produce extremely concise tableau rules for classical as well as for many-valued logics, including the otherwise difficult to handle Łukasiewicz sum.

For classical propositional logic the rules summarized in Table 3 constitute a sound and complete rule set. The variables in the signs run over $\{0, 1\}$. In the case of many-valued logics they would run over N.

It can be easily shown that for each instance of the variables that solve the annotated integer program (IP), a rule that is sound in the usual sense (that is, it preserves satisfiability) results. On the other hand, the conclusions of all instances of a rule with a solved annotated IP together form a complete set of rule extensions in the usual sense (that is, one of them preserves unsatisfiability).

Regarding branch closure we may view atomic formulas (=propositional variables) as object variables ranging over the set of truth values. We can take advantage of the fact that the (meta-)variables in the signs and the (object-)variables in the formulas are of the same type and merge them into a single constraint. Specifically, if p is atomic and $\boxed{\geq i}\ p$ is present on the current (and only) branch we simply add the constraint $p \geq i$ to the IP already associated with the branch, and we add the constraint $q \leq j$, when $\boxed{\leq j}\ q$ is present. Branch closure is then encoded in the fact that all generated constraints cannot be solved simultaneously.

Note that all rules are linear, so we can extract from the fully expanded tableau together with the closure conditions a single IP problem whose number of variables is not greater than the length of the input formula. There is ample room for improvements of various sorts. For example, two of the rules in Table 3 can be improved in the following way:

$$\frac{\boxed{\leq i}\ \alpha}{\begin{array}{c}\boxed{\leq i-j+1}\ \alpha_1 \\ \boxed{\leq j}\ \alpha_2\end{array}} \ \ i \leq j \qquad\qquad \frac{\boxed{\geq i}\ \beta}{\begin{array}{c}\boxed{\geq i-j}\ \beta_1 \\ \boxed{\geq j}\ \beta_2\end{array}} \ \ i \geq j$$

Table 3: Classical Tableau Rules in Constraint Formulation.

$$\frac{\boxed{\leq i}\,\alpha}{\begin{array}{l}\boxed{\leq i_1}\,\alpha_1 \\ \boxed{\leq i_2}\,\alpha_2\end{array}}\; i_1+i_2 \leq i+1 \qquad \frac{\boxed{\geq i}\,\alpha}{\begin{array}{l}\boxed{\geq i}\,\alpha_1 \\ \boxed{\geq i}\,\alpha_2\end{array}} \qquad \frac{\boxed{\geq i}\,\beta}{\begin{array}{l}\boxed{\geq i_1}\,\beta_1 \\ \boxed{\geq i_2}\,\beta_2\end{array}}\; i_1+i_2 \geq i \qquad \frac{\boxed{\leq i}\,\beta}{\begin{array}{l}\boxed{\leq i}\,\beta_1 \\ \boxed{\leq i}\,\beta_2\end{array}} \qquad \frac{S\,\neg\neg\phi}{S\,\phi}$$

In the conclusion only one new variable is introduced instead of two. The price is to admit linear expressions in the signs, but that is no problem.

As said before, this approach works well also for finitely-valued logics and even for some ∞-valued logics. In the case of ∞-valued logics, some of the variables are over the rationals, some are binary. Let us give a linear constraint rule formulation for the signed formula $\boxed{\leq i}\,\phi \oplus \psi$ in the ∞-valued case. If we plot the three-dimensional region which is spanned by the triples (ϕ, ψ, i) for which $\boxed{\leq i}\,\phi \oplus \psi$ is true (the hypograph of $\oplus$) we obtain the union of the two regions depicted in Figure 2.

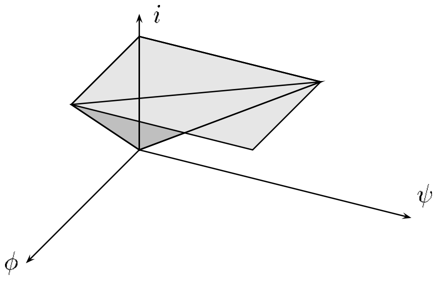


Figure 2: Hypograph of $\oplus$ in ∞-valued logic.

The resulting region is not convex, so it cannot be represented as a linear program. We can, however, introduce a new binary variable and represent it conveniently with a mixed $0-1$ program resulting in the tableau rule shown in Table 4.

Table 4: Linear rule for $\boxed{\leq i}$ and $\oplus$ using MIP constraints.

$$\frac{\boxed{\leq i}\,\phi \oplus \psi}{\begin{array}{ll}\boxed{\leq i_\phi}\,\phi & y \leq i,\ i_\phi + i_\psi \leq y + i \\ \boxed{\leq i_\psi}\,\psi & y \leq i_\phi,\ y \leq i_\psi\end{array}}$$

There, i_ϕ and i_ψ are rational variables, while y is a binary **control variable** which selects the lighter shadowed convex region if $y = 1$, the (partly hidden) darker shadowed convex region if $y = 0$.

The tableau-based translation from formulas to integer problems can be extended to the first-order case with appropriate free variable quantifier rules [10, 3]. Two main problems have to be addressed then: first, as usual in semantic tableaux, one does not know beforehand how many copies of a γ formula (corresponding to the number of different instances in Herbrand's theorem) are needed in order to get a refutation. Second, the resulting integer programs from such a translation contain free first-order variables which have to be instantiated somehow. On the other hand, similar problems have to be addressed in most theorem proving systems and it may have certain advantages to use integer programming as a ground solver besides the capability to handle many-valued logic. See [26] for a more detailed account of first-order issues and [14, 15] on integer programming methods for many-valued logic.

7 Implementation

What recommendations concerning the logical basis can we give to somebody who is willing to build a many-valued theorem prover for real applications?

The answer strongly depends on the expressive power of the logics under consideration and, to a certain degree, on the intended application.

For propositional ∞-valued logics an MIP-based implementation as sketched in Section 6 is probably best. For other logics, the state-of-the-art approach in the two-valued case should be taken and modified using the sets-as-signs technique. In particular, for propositional finitely-valued logics a modification of the Davis-Putnam procedure seems best if tautology checking is the aim [5] and decision diagrams if simplification of large and unstructured expressions is desired [4]. For first-order finitely-valued logics a resolution framework as sketched in Section 5 is very interesting. In the future also refinements of tableaux might turn out to be competitive [22] or perhaps a tableau-based approach incorporating integer programming [26].

For the time being two tools are under development at University of Karlsruhe. The tableau-based many-valued theorem prover[7] ${}_3T^AP$ [17, 2] can in principle handle arbitrary finitely-valued first-order logics and makes use of sets-as-signs, regularity properties and optimized standard quantifier rules. In addition it incorporates other state-of-the-art features of classical tableau-based provers such as lemma generation (also for many-valued logic, see [15]) and unification [3].

To gain some insight into how much is achieved in practice with the sets-as-signs concept, two implementations of the three-valued first-order logic used in [11] were provided. The first version runs with rules which use only singleton signs. The second version has a full set of signs and rules, that is, for each $\emptyset \subsetneq S \subsetneq \{0, \frac{1}{2}, 1\}$ and each connective a rule is computed. In Table 5 statistical figures of runs in both versions are summarized for various first-order problems from [11].

Run times are roughly proportional to the number of generated branches, thus there is a very clear advantage for the sets-as-signs approach. All run times required in this case are within fractions of a second. Note that the sample problems are not particularly hard and the underlying logic has only three truth values. For logics with a larger number of truth values the difference becomes even more spectacular.

The second tool is a mixed integer programming implementation (written in C^{++}) together with a transformation algorithm from logic into MIP along the lines in Section 6 (written in PROLOG). The performance for classical propositional logic is not quite as good as state-of-the-art satisfiability checkers [5], but the system is still under development (including the extension to first-order logic) and moreover works for all finitely-valued and most ∞-valued logics. We hope to put a first version of this system into the public domain during this year.

For prototyping purposes, a constraint logic programming (CLP) approach offers an easy way to implement finitely-valued and infinitely-valued logics. Consider the rule from Table 4. It can be written more compactly as the top left rule in Table 6. To the right and below of it the rules for $\boxed{\geq i}$ and for negation are shown.

These rules can be translated one to one into the following constraint logic program (in $CLP(\mathcal{R})$ syntax, cf. [19]):

[7] ${}_3T^AP$ is available without charge to research institutions. Please contact the author if you are interested in receiving a copy. ${}_3T^AP$ is written in PROLOG and runs on Quintus Prolog 3.1 and higher and Sicstus Prolog 2.1.8 and higher.

Table 6: Improved rules for $\oplus$ and negation.

$$\frac{\boxed{\leq i}\ \phi \oplus \psi}{\begin{array}{ll} \boxed{\leq i-j+y}\ \phi & y \leq i \\ \boxed{\leq j+y}\ \psi & j \leq i \end{array}} \qquad \frac{\boxed{\geq i}\ \phi \oplus \psi}{\begin{array}{l} \boxed{\geq i-j}\ \phi \\ \boxed{\geq j}\ \psi \end{array}}$$

$$\frac{\boxed{\geq i}\ \neg\phi}{\boxed{\leq 1-i}\ \phi} \qquad \frac{\boxed{\leq i}\ \neg\phi}{\boxed{\geq 1-i}\ \phi}$$

```
leq(plus(Phi,Psi),I) :- leq(Phi,I-J+Y),
                        leq(Psi,J+Y),
                        truth_var(J),
                        control_var(Y),
                        Y<=I, J<=I.

leq(neg(Phi),I)      :- geq(Phi,1-I).

leq(atom(Phi),I)     :- truth_value(Phi),
                        Phi<=I.

geq(plus(Phi,Psi),I) :- geq(Phi,I-J),
                        geq(Psi,J),
                        truth_var(J).

geq(neg(Phi),I)      :- leq(Phi,1-I).

geq(atom(Phi),I)     :- truth_value(Phi),
                        I<=Phi.

control_var(0).
control_var(1).

truth_var(J)         :- 0<=J, J<=1.
```

In order to test, for instance, the formula $\neg p \oplus p$ for validity, it is sufficient to verify that the following query fails:

```
?- Phi = plus(neg(atom(P)),atom(P)),
   leq(Phi,C), truth_var(C), C<1.
```

To avoid operator definitions, we write the input as a PROLOG term, where the unary function `atom` denotes that its argument is an atomic formula. If the query fails, then the truth value of `Phi` cannot be smaller than 1 under any valuation, hence, `Phi` must be a $\{1\}$-tautology of the infinitely-valued logic over $\oplus$ and $\neg$. We assume, of course, that strict inequality constraints are implemented properly (otherwise, we have to minimize C, and we can no longer use CLP, but only a proper MIP implementation).

The same program can also be used for finding satisfying valuations. If `Phi` is $\{1\}$-satisfiable, then the query

Table 5: Some Test Results with $_3T^AP$.

Problem	Closed branches when using			
	singleton signs		Sets-as-Signs	
	Unlinked[8]	Linked[9]	Unlinked	Linked
Lemma 5.1	225	119	7	7
Theorem 5.3	225	82	23	13
Lemma 5.9	—[10]	33	4	4
Figure 5.14	254	254	10	8
Figure 5.17	—	—	—	8
Axiom MVEQ4	46	33	6	4

```
?- Phi = ..., leq(Phi,1), geq(Phi,1).
```

yields a satisfiable constraint system over the variables occurring in Phi, such that every solution to this system forces Phi to evaluate to 1, in other words, the solution $\{1\}$-satisfies Phi.

We obtain a theorem prover for n-valued logic simply by changing the definition of the predicate truth_var/1 as follows:

```
truth_var(0).
truth_var(1/(<n>-1)).
...
truth_var(1).
```

Finally, we would like to mention that [24] report an experimental resolution prover for finitely-valued logics that operates on signed clauses.

8 Conclusion and Outlook

In this paper we have reviewed some recently developed techniques for automated deduction in first-order and propositional many-valued logics. Where are the main prospects for future research that can lead to further improvements and new applications?

The computational properties of many-valued logics are still only understood in special cases. Tools like the translation technique to MIP could be used to identify classes of logics with interesting computational properties as, for instance, the existence of strong cutting planes for the resulting MIP [20].

Another interesting topic is the development of resolution or tableau closure rules that can exploit ordering of the truth values to prune the search space. This could be done by generalizing sets-as-signs to ordered-sets-as-signs as in Definition 12. Exploiting the order could then lead either to more concise rules (in the tableau setting) or to less resolvents (in the resolution setting). Looking at up- and downsets in truth value lattices seems to be a good starting place. See also [9] for a related approach in the domain of substructural logics.

Finally, the upsets and downsets occurring in the MIP translation can be considered as a natural generalization of positive and negative formulas in the sense of logic programming. Hence, a constraint tableau rule might be interpreted as (a) clause(s) of a constraint logic program (with linear arithmetic constraints). This line of thought would lead to many-valued analoga of Horn formulas, definite Horn formulas, etc. Also there are connections to the work of Subrahmanian *et al.* on implementing non-monotonic reasoning with constraint logic programs using linear arithmetic constraints [21] which have not been explored yet. In this context it is also interesting to note that there has recently been established a close relationship between many-valued logics and non-monotonic logics [28].

[8] All formulas are present on the initial branch.
[9] Only formulas with links into the formula in focus are fetched from the knowledge base on demand.
[10] No proof found after several minutes.

References

[1] M. Baaz and C. G. Fermüller. Resolution for many-valued logics. In A. Voronkov, editor, *Proc. Logic Programming and Automated Reasoning LPAR'92*, pages 107–118. Springer, LNAI 624, 1992.

[2] B. Beckert, S. Gerberding, R. Hähnle, and W. Kernig. The many-valued tableau-based theorem prover $_3T^AP$. In D. Kapur, editor, *Proc. 11th Conference on Automated Deduction, Albany/NY*, pages 758–760. Springer LNAI 607, 1992.

[3] B. Beckert, R. Hähnle, and P. H. Schmitt. The *even more* liberalized δ-rule in free variable semantic tableaux. In G. Gottlob, A. Leitsch, and D. Mundici,

editors, *Proceedings of the third Kurt Gödel Colloquium KGC'93, Brno, Czech Republic*, pages 108–119. Springer LNCS 713, Aug. 1993.

[4] R. Y. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical report, Carnegie Mellon University. School of Computer Science, 1992.

[5] M. Buro and H. K. Büning. Report on a SAT competition. Reihe Informatik 110, FB 17—Mathematik/Informatik, Universität Paderborn, Nov 1992.

[6] W. A. Carnielli. Systematization of finite many-valued logics through the method of tableaux. *Journal of Symbolic Logic*, 52(2):473–493, June 1987.

[7] W. A. Carnielli. On sequents and tableaux for many-valued logics. *Journal of Non-Classical Logic*, 8(1):59–76, May 1991.

[8] J. Czelakowski. Algebraizability of logic and the deduction theorem. Lecture Notes Fourth Summer School on Logic, Language and Information, Colchester/England, Aug. 1992.

[9] M. D'Agostino and D. M. Gabbay. Labelled refutation sustems: A case study. In *Proc. Second Workshop on Theorem Proving with Tableaux and Related Methods, Marseille.* Technical Report, MPI Saarbrücken, 1993.

[10] M. C. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, New York, 1990.

[11] S. Gerberding. Monomorphe Axiomatisierung von Intervallarithmetiken mit mehrwertigen Logiken. Master's thesis, University of Karlsruhe, Department of Computer Science, December 1991.

[12] R. Hähnle. Towards an efficient tableau proof procedure for multiple-valued logics. In *Proceedings Workshop on Computer Science Logic, Heidelberg*, pages 248–260. Springer, LNCS 533, 1990.

[13] R. Hähnle. Uniform notation of tableaux rules for multiple-valued logics. In *Proc. International Symposium on Multiple-Valued Logic, Victoria*, pages 238–245. IEEE Press, 1991.

[14] R. Hähnle. A new translation from deduction into integer programming. In J. Calmet and J. A. Campbell, editors, *Proc. Int. Conf. on Artificial Intelligence and Symbolic Mathematical Computing AISMC-1, Karlsruhe, Germany*, pages 262–275. Springer, LNCS 737, 1992.

[15] R. Hähnle. *Automated Theorem Proving in Multiple-Valued Logics*, volume 10 of *International Series of Monographs on Computer Science.* Oxford University Press, 1993.

[16] R. Hähnle. Short conjunctive normal forms in finitely-valued logics. *Journal of Logic and Computation, to appear*, 1994.

[17] R. Hähnle, B. Beckert, S. Gerberding, and W. Kernig. The Many-Valued Tableau-Based Theorem Prover $_3T^AP$. IWBS Report 227, Wissenschaftliches Zentrum Heidelberg, IWBS, IBM Deutschland, July 1992.

[18] R. Hähnle and W. Kernig. Verification of switch level designs with many-valued logic. In A. Voronkov, editor, *Proc. LPAR'93, St. Petersburg*, pages 158–169. Springer, LNAI 698, 1993.

[19] N. Heintze, J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. *The CLP($\mathcal{R}$) Programmer's Manual, Version 1.2.* IBM Watson Research Center, Yorktown Heights, Sept. 1992.

[20] J. N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4:45–69, 1988.

[21] V. Kagan, A. Nerode, and V. S. Subrahmanian. Computing definite logic programs by partial instantiation and linear programming, 1993.

[22] R. Letz, J. Schumann, S. Bayerl, and W. Bibel. SETHEO: A high-perfomance theorem prover. Technical report, Forschungsgruppe Künstliche Intelligenz, TU München, 1991.

[23] J. J. Lu, L. J. Henschen, V. Subrahmanian, and N. C. A. da Costa. Reasoning in paraconsistent logics. In R. Boyer, editor, *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 181–210. Kluwer, 1991.

[24] J. J. Lu, N. V. Murray, and E. Rosenthal. Signed formulas and annotated logics. In *Proc. International Symposium on Multiple-Valued Logics*, pages 48–53, 1993.

[25] N. V. Murray and E. Rosenthal. Signed formulas: A liftable meta logic for multiple-valued logics. In *Proceedings ISMIS'93, Trondheim, Norway*, pages 275–284. Springer LNCS 689, 1993.

[26] K. Ries and R. Hähnle. Prädikatenlogisches Beweisen mit gemischt ganzzahliger Optimierung. Ein tableaubasierter Ansatz. In *Working Notes of Workshop Künstliche Intelligenz und Operations Research, Berlin (published as Tech Report, Max-Planck-Institut für Informatik, Saarbrücken, MPI-I-93-???)*, 1993.

[27] R. Smullyan. *First-Order Logic.* Springer, New York, 1968.

[28] Z. Stachniak. Algebraic semantics for cumulative inference operations. In *Proc. AAAI'93, Raleigh/NC*, 1993.

[29] W. Suchoń. La méthode de Smullyan de construire le calcul n-valent de Lukasiewicz avec implication et négation. *Reports on Mathematical Logic, Universities of Cracow and Katowice*, 2:37–42, 1974.

[30] G. Tseitin. On the complexity of proofs in propositional logics. *Seminars in Mathematics*, 8, 1970. Reprinted in: Siekmann, J. and Wrightson G. (eds.) *Automation of Reasoning 2: Classical Papers on Computational Logic*, pp. 466–483, Springer, 1983.