A New Translation from Deduction into Integer Programming

Reiner Hähnle*

Institut für Logik, Komplexität und Deduktionssysteme
Fakultät für Informatik, Universität Karlsruhe Kaiserstr. 12,
7500 Karlsruhe, Germany
haehnle@ira.uka.de

Abstract

We generalize propositional analytic tableaux for classical and many–valued logics to *constraint tableaux*. We show that this technique provides a new translation from deduction into integer programming. The main advantages are (i) an efficient satisfiability checking procedure for classical and, for the first time, for a wide range of many–valued, including infinitely–valued propositional logics; (ii) a new point of view on classifying complexity of many–valued logics; (iii) easy NP–containment proofs for many–valued logics.

Keywords: Mechanical Theorem Proving, Integer Programming, Linear Programming, Fuzzy Reasoning, Verification of Integrated Circuits

Introduction

We assume the reader is familiar with tableau calculus for classical propositional logic. Excellent introductions are [1, 2]. We assume further some basic knowledge of propositional many–valued logics [3]. Due to space limitations no proofs are included; most of them can be found in [4].

We consider a finitely–valued logic $\mathcal{L}$ consisting of a propositional language $\mathbf{L}$ and a n–valued matrix $\mathbf{A}$. As the set N of truth values we take equidistant rational numbers from the unit interval i.e.

$$N = \{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\}$$

In a three–valued logic, for example, $N = \{0, \frac{1}{2}, 1\}$. We fix the set of designated truth values (i.e. the truth values that support validity of a statement) as $\{1\}$. The semantic matrix of a logic $\mathcal{L}$ may be given by many–valued truth tables for each logical connective. For example, we might define so–called three–valued Kleene disjunction with the truth table shown on the left in Figure 1.

While in classical signed tableau calculus the signs are either 1 or 0 and thus correspond exactly to a classical truth value, the author has introduced in [5] arbitrary subsets

*The research described in this paper has been conducted while the author was sponsored from IBM Germany.

of the truth value set as signs. As an example we give the tableau rule corresponding to the sign $\{\frac{1}{2}, 1\}$ and three–valued Kleene disjunction (see Figure 1).

$\vee$	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

$$\frac{\{\frac{1}{2}, 1\}\ \phi \vee \psi}{\{\frac{1}{2}, 1\}\ \phi \mid \{\frac{1}{2}, 1\}\ \psi}$$

Figure 1: Truth Table and Tableau Rule for Kleene Disjunction.

The truth value of the formula in the premise is *undefined* or *true* if and only if the truth value of one of its direct subformulas is *undefined* or *true*.

In [6] it has been shown that for a non–trivial class of many–valued logics Smullyan style tableau systems[1] result if this approach is being used.

On the other hand, for some important classes of many–valued logics, in particular for Łukasiewicz logics (which are defined below), this approach is not completely satisfying, since the branching factor of rules can become as big as the number of truth values. Moreover, the method does not extend to infinitely–valued logics.

For this reason, we develop in the following section the concept of a constraint tableau rule which is then used to construct a compact tableau system for Łukasiewicz (and many other) logics.

Tableau Proofs with Constraints

As mentioned before, the signs used in our tableau systems correspond to subsets of the set of truth values. We will, however, not admit arbitrary truth value sets as signs, but only signs of a certain shape (cf. [6]). We define the following abbreviations:

$$\begin{aligned} \boxed{\leq i} &:= \{0, \ldots, i\} \text{ for } i \in N \\ \boxed{\geq i} &:= \{i, \ldots, 1\} \text{ for } i \in N \end{aligned}$$

We do not impose any restrictions on the connectives.

Consider a signed formula $\phi = \boxed{\geq i} F(\phi_1, \phi_2)$, where F is any 2–place connective. A signed formula of this type is satisfiable iff for some valuation v the value of $v(F(\phi_1, \phi_2))$ is greater or equal than i. The key idea in the following is to leave i as well as the signs in the rule extensions uninstantiated. For example, we could write down a rule like

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

For most instances of i, i_1, i_2, however, such a rule does not properly reflect the semantics of F, hence we must impose some additional constraints. Let us become a little bit

[1] That is, the rules may be classified according to Smullyan's uniform notation [1] into type $\alpha, \beta, \gamma, \delta$.

more concrete and consider a signed formula $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$, where $\supset_{\text{Ł}}$ denotes n–valued Łukasiewicz implication which can be defined as

$$i_1 \supset_{\text{Ł}} i_2 = \min\{1, 1 - i_1 + i_2\} \qquad \text{or, for } n = 3:$$

$\supset_{\text{Ł}}$	0	$\frac{1}{2}$	1
0	1	1	1
$\frac{1}{2}$	$\frac{1}{2}$	1	1
1	0	$\frac{1}{2}$	1

If $i = 1$ the signed formula $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ is trivially satisfied and can be omitted from the further analysis of the current branch, otherwise we have

Proposition 1 *If $i < 1$ or, equivalently, $i \leq \frac{n-2}{n-1}$ then $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ is satisfiable iff both, $\boxed{\geq i_1}\phi_1$ and $\boxed{\leq i_2}\phi_2$ are satisfied by the same valuation and $i = 1 - i_1 + i_2$ holds.*

From this proposition we may derive a tableau rule for $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$. It has *provisos* which have to be satisfied in any proof the rule is used in. A similar technique is used in the classical quantifier rules, only that the proviso can be checked immediately in the case of quantifier rules, whereas we delay the check until tableau completion in the present case. Moreover, there may be different constraints associated with each extension. Let us call rules of the new kind **constraint rules**. The rule for $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ and its counterpart for $\boxed{\geq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ are given in Table 1. Note that the left extension of the rule for $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ is empty; only the constraint information $i = 1$ counts in the corresponding branch.

$$\frac{\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{l|l} & \boxed{\geq i_1}\phi_1 \quad i \leq \frac{n-2}{n-1} \\ i = 1 & \boxed{\leq i_2}\phi_2 \quad 1 - i_1 + i_2 = i \end{array}} \qquad \frac{\boxed{\geq i}(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{l} \boxed{\leq i_1}\phi_1 \\ \boxed{\geq i_2}\phi_2 \quad 1 - i_1 + i_2 = i \end{array}}$$

Table 1: Constraint Rules for $\boxed{\leq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$ and $\boxed{\geq i}(\phi_1 \supset_{\text{Ł}} \phi_2)$.

Recall that the number of extensions for n–valued Łukasiewicz implication rules was up to n with the old rules, while now it is constant for arbitrary n. Different values of n are handled in the constraints. Thus, a proof tree looks the same for the same root formula and different n.

It is instructive to instantiate the premise of a rule and compute all solutions of its constraint system. Consider, for example, the rule for $\boxed{\geq \frac{1}{4}}(\phi_1 \supset_{\text{Ł}} \phi_2)$ in 5–valued Łukasiewicz logic. The constraint system consists of the single equation $1 - i_1 + i_2 = \frac{1}{4}$. The values of the $(i_1, i_2) \in N^2 = \{0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1\}^2$ solving this equation are $\{(\frac{3}{4}, 0), (1, \frac{1}{4})\}$.

Each different solution of the constraint system gives a conventional rule extension, so we can back–translate our example into a conventional rule with two extensions and no constraints:

$$\frac{\boxed{\geq \frac{1}{4}}(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{l|l} \boxed{\leq \frac{3}{4}}\phi_1 & \boxed{\leq 1}\phi_1 \\ \boxed{\geq 0}\phi_2 & \boxed{\geq \frac{1}{4}}\phi_2 \end{array}}$$

Eliminating the trivially satisfiable formulas $\boxed{\geq 0}\,\phi_2$ and $\boxed{\leq 1}\,\phi_1$ yields exactly the same rule for $\boxed{\geq \frac{1}{4}}\,(\phi_1 \supset_{\mathbf{L}} \phi_2)$ as the method in [5]. The constraint $1 - i_1 + i_2 = \frac{1}{4}$ is merely a compact representation of the extensions in the conventional rule.

We postpone the question of branch closures involving signs of the new kind for the moment and observe that with each branch of a completed tableau which is not yet tested for closure a system of linear inequalities whose variables are ranging over N, in other words an *Integer Programming* (IP) problem with solutions in N, is associated.[2]

We modify the usual *tableau construction process* as follows:

> When a constraint rule is applied, the constraints from each extension are added to the constraints already present on the current branch and thus form a new constraint system associated with each newly generated branch. It may happen that for one or more of the newly generated branches the new constraint system becomes unsolvable. Such branches are deleted immediately, since they cannot possibly represent a satisfiable extension of the current branch.

In general, this is how a constraint tableau rule does look like:

$$\begin{array}{c|c|c} \multicolumn{3}{c}{\mathsf{S}\,\phi} \\ \hline C_1 & \cdots & C_r \\ (IP_1) & & (IP_r) \end{array} \qquad \text{where} \qquad \begin{array}{rcl} \phi & = & F(\phi_1, \ldots, \phi_k) \\ S & = & \boxed{\lesseqgtr i} \\ C_j & = & \boxed{\lesseqgtr i_{j1}}\,\phi_1 \circ \cdots \circ \boxed{\lesseqgtr i_{jk}}\,\phi_k \\ IP_j & = & A_j I_j \leq B_j \\ I_j & = & i, i_{j1}, \ldots, i_{jk} \end{array}$$

Together with the specification of a general constraint rule we must prove an analogue to Proposition 1:

Proposition 2 (Constraint Rule Adequacy) *Using the notation from above we have that* $\mathsf{S}\phi$ *is satisfiable iff for some instantiation of the variables in* I_j *in at least one extension (i)* IP_j *is solved and (ii)* C_j *is satisfiable.*

With the following definition one may link conventional tableaux and tableaux constructed with the help of constraint rules:

Definition 3 (Partial Tableau) *Let* $\mathbf{T}$ *be a completed tableau for a set of formulas* Φ *constructed with the help of constraint rules. Let* I_l *be the set of variables occurring in the IP problem associated with a branch* $\mathbf{B}_l$ *of* $\mathbf{T}$*. If all variables in all* I_l *are instantiated with values from* N *such that all IP problems are solved we call the resulting tree* $\mathbf{T}^*$ **partial tableau** *for* Φ*.*

The idea is that each solution instance of a constraint tableau corresponds to the partial description of a conventional tableau. The collection of completed partial tableaux for Φ determines a certain completed conventional tableau for Φ.

[2] Strictly speaking, this is only the *feasability* part of an IP problem. Nevertheless we speak always of IP problems in the following, although we never need to maximize a cost function.

Based on these concepts one may derive suitable definitions of Hintikka set and Analytic Consistency Property involving partial tableaux. For these Hintikka's Lemma and a Model Existence Theorem can be proved in rather the same way as in [5].

If we are interested in IP problems the constraints generated by the rules have to be linear inequalities. Later we will investigate the question for which class of connectives such constraints allow substantial simplifications of rules. For the moment, it is sufficient to note that all many–valued connectives principally fit into the new framework, since the old rules are just a special case using the empty constraint.

Intuitively, a formula is valid when every partial tableau of its negation can be closed. Let us see how we can deal with tableau closure in the present setting.

First, we note that in a completed tableau it is sufficient to look for atomic closure. Also it is sufficient to look for *pairs* of contradictory formulas (instead of tupels) when signs are defined as above. Moreover, closure can only occur between atomic formulas with a different type of sign, that is, it can never occur between formulas like $\boxed{\leq i_1}\,\phi$ and $\boxed{\leq i_2}\,\phi$. Thus, a branch can only be closed when either two atomic formulas $\boxed{\leq i_1}\,\phi$, $\boxed{\geq i_2}\,\phi$ are present or a single formula $\boxed{\leq j}\,\psi$ (or $\boxed{\geq j}\,\psi$) such that no rule is defined for some j.[3]

In the first case, the branch is closed iff the signs have an empty intersection iff $i_1 < i_2$. In the second case, consider $\boxed{\leq j}\,\psi$. There must be a greatest j_0 such that no rule for $\boxed{\leq j_0}\,\psi$ is defined. Then, obviously no rule for $\boxed{\leq j}\,\psi$ is defined iff $j \leq j_0$ iff $j < j_0 + \frac{1}{n-1}$. The value of j_0 is easily obtained from the truth table of the top level connective of ψ. For $\boxed{\geq j}\,\psi$ we proceed similar.

We obtain thus for each tableau branch $\mathbf{B}_l$ a set of strict linear inequalities

$$\{c_{l_1} i_{l_1} < d_{l_1}, \ldots c_{l_p} i_{l_p} < d_{l_p}\},$$

such that solving *any* of them results in the closure of $\mathbf{B}_l$, in other words, a disjunction of strict linear inequalities. By definition, a branch is open when it cannot be closed. If we negate the disjunction

$$\bigvee_{k=1}^{p} (c_{l_k} i_{l_k} < d_{l_k}),$$

apply DeMorgan's law and observe that $c_{l_k} i_{l_k} \not< d_{l_k}$ iff $c_{l_k} i_{l_k} \geq d_{l_k}$ we get

Definition 4 *Let* $\mathbf{T}$ *be a completed tableau for* Φ *built up using constraint rules and let* $\mathbf{B}_1, \ldots, \mathbf{B}_m$ *be the branches of* $\mathbf{T}$. *Moreover, let* $C_l I_l \leq D_l$ *be the IP problem (that is,* $\bigwedge_{k=1}^{p} c_{l_k} i_{l_k} \geq d_{l_k}$*) corresponding to the closure of* $\mathbf{B}_l$ *as above. Then* $\mathbf{B}_l$ *is called* **open** *iff* $C_l I_l \leq D_l$ *has a solution that solves also the IP problem* $A_l I_l \leq B_l$ *associated with the rule applications on* $\mathbf{B}_l$. $\mathbf{T}$ *is a* **constraint tableau proof** *of* Φ *iff it has no open branch.*

Actually, there is another, in some sense simpler way to represent branch closure. If we view atomic formulas (that is, propositional variables) as object variables ranging over the set of truth values we can take advantage from the fact that the (meta) variables in the signs and (object) variables are of the same type and mix them together in a

[3] If no truth value in $\boxed{\leq j}$ does occur in the truth table of a connective F, a signed formula $\boxed{\leq j}\,F(\phi, \psi)$ is unsatisfiable and no corresponding tableau rule is defined. We call such formulas *self–contradictory*.

single constraint. If p is atomic and $\boxed{\geq i}\, p$ is present on the branch $\mathbf{B}_l$ we simply add the constraint $p \geq i$ and we do similar for $\boxed{\leq i}\, p$. The resulting constraint system on a branch has then to be solved over $I_l \cup P_l$, where P_l are the propositional variables occurring on $\mathbf{B}_l$. This representation has the advantage of being shorter than the one without object variables, but it has the disadvantage of involving a greater number of variables.[4]

Theorem 5 *ϕ is a tautology iff there is a completed tableau for $\boxed{\leq \frac{n-2}{n-1}}\, \phi$ built up using constraint rules which represents a constraint tableau proof.*

Example

Before we proceed, let us give an example of a tableau proof using constraint rules. We will give a proof of the formula $p \supset_{\mathrm{L}} (q \supset_{\mathrm{L}} p)$ in three–valued Łukasiewicz logic. In the upper part of Figure 2 the conventional proof is shown, in the lower part the proof tree using constraint rules. Note that rule application to formula (3) in the lower tree does yield only one branch, since adding the condition $i_2 = 1$ would make the constraint system unsolvable.

The following matrices correspond to the only branch of the tree on the bottom (note that an equality is represented by two inequalities):

$$A = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & -1 \\ 0 & -1 & -1 & 1 \end{pmatrix}, B = \begin{pmatrix} -\frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \\ 1 \\ -1 \end{pmatrix}, C = \begin{pmatrix} 1 & 0 & 0 & -1 \end{pmatrix}, D = \begin{pmatrix} 0 \end{pmatrix}, I = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{pmatrix}$$

$$\text{We must show that } \begin{pmatrix} A \\ C \end{pmatrix} I \leq \begin{pmatrix} B \\ D \end{pmatrix} \text{ is not solvable over } N = \{0, \frac{1}{2}, 1\}.$$

This may seem not a great achievement if compared to constructing the conventional proof tree, but we must take into account that there exist very efficient algorithms for solving IP problems and, more important, while the IP problem *does not become substantially more complex when n grows*, the conventional proof tree becomes bigger and bigger (in the example it grows with $\mathcal{O}(n^2)$, but in general it grows with $\mathcal{O}(n^k)$, where k is the depth of the formula to be proved).

Another important point is that the system of inequalities for each branch can be computed incrementally, while the tree is constructed, thus using information that was computed only once in more than one branch.

Infinitely Valued Logic

As the proof trees for all finite Łukasiewicz logics look the same modulo some constants, it is a natural question to ask, whether the method can be generalized to *infinitely valued* logic. The answer is in the affirmative and all we must do is to handle strict inequalities

[4] This phenomenon is characteristic for representation theory of linear integer constraints, see [7, p. 8].

as signs such as in $\boxed{<i}$. As a consequence, also in the constraints strict inequalities will occur. In Table 2 we have summarized the rules for infinitely valued Łukasiewicz logic $\mathrm{Ł}_\omega$, while in Figure 3 we give the infinitely valued version of the example from above. An analogue to Proposition 1 can be used to show soundness and completeness of that system.

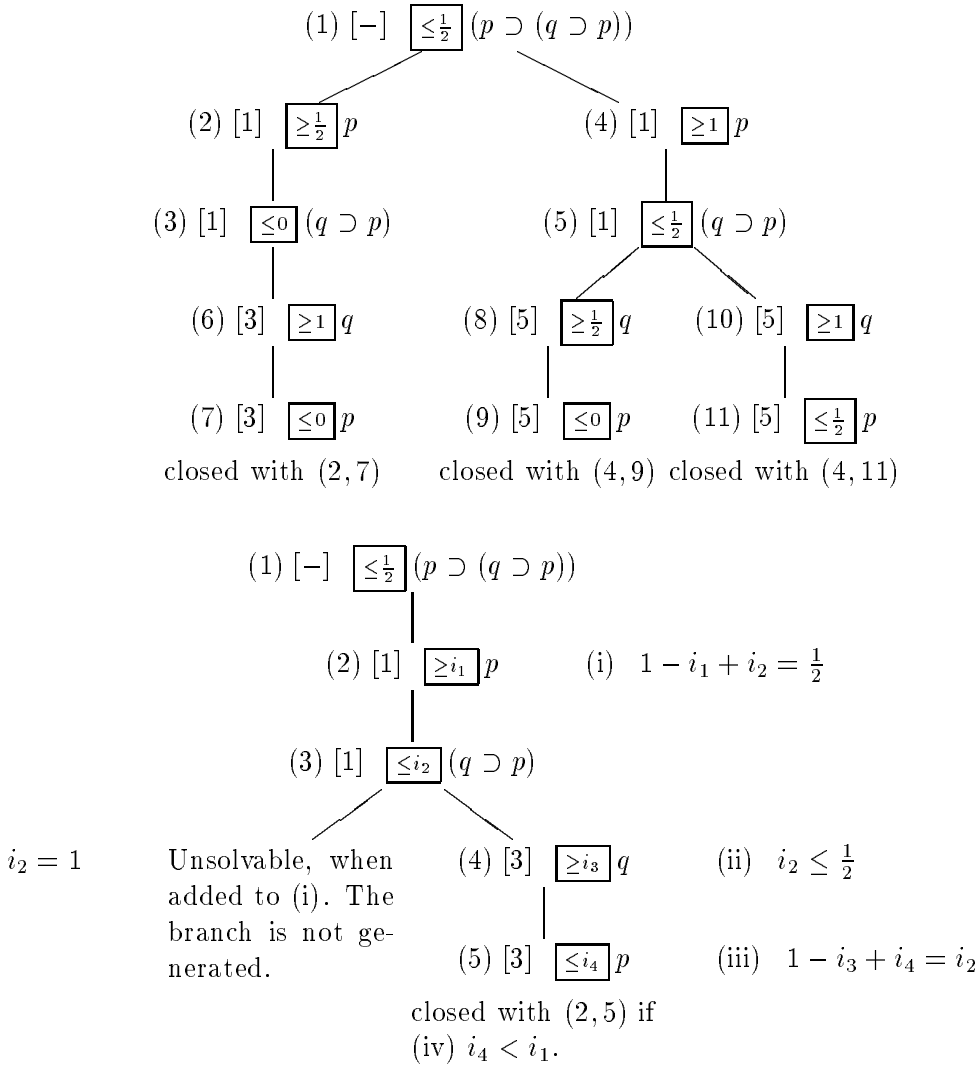

Figure 2: A Simple Derivation in Three–Valued Łukasiewicz Logic with and without Constraints.

Note that if we work in infinitely valued logic we must try to solve the constraint systems over the unit interval of the rational line and thus have *Linear Programming* (LP) problems instead of IP problems. Since LP is known to be in P we have polynomial satisfiability in infinitely–valued Łukasiewicz logic for all formulas whose tableau proof trees branch at most polynomially often.

The signs $\boxed{<1}$ and $\boxed{>1}$ occur only in the initial tableau.

$$\frac{\boxed{\leq i}\,(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{l|ll} & \boxed{\geq i_1}\,\phi_1 & i < 1 \\ i = 1 & \boxed{\leq i_2}\,\phi_2 & 1 - i_1 + i_2 = i \end{array}} \qquad \frac{\boxed{\geq i}\,(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{ll} \boxed{\leq i_1}\,\phi_1 & \\ \boxed{\geq i_2}\,\phi_2 & 1 - i_1 + i_2 = i \end{array}}$$

$$\frac{\boxed{<i}\,(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{ll} \boxed{\geq i_1}\,\phi_1 & \\ \boxed{\leq i_2}\,\phi_2 & 1 - i_1 + i_2 < i \end{array}} \qquad \frac{\boxed{>i}\,(\phi_1 \supset_{\text{Ł}} \phi_2)}{\begin{array}{ll} \boxed{\leq i_1}\,\phi_1 & \\ \boxed{\geq i_2}\,\phi_2 & 1 - i_1 + i_2 < i \end{array}}$$

Table 2: Constraint Rules for Infinitely Valued Łukasiewicz Logic.

Complexity of Many–Valued Logics

At this point we can ask ourselves which classes of many–valued logics can be axiomatized naturally using constraint tableau rules. Consider the $k+1$–dimensional region R that is defined by each combination of sign variable and k–ary connective f. Let $\mathsf{S}(i)$ be any of $\boxed{<i}$, $\boxed{>i}$, $\boxed{\leq i}$, $\boxed{\geq i}$ or $\{\mathrm{i}\}$. Define

$$R_f(\mathsf{S}(i)) = \{(i, i_1, \ldots, i_k) |\, f(i_1, \ldots, i_k) \in \mathsf{S}(i); i, i_1, \ldots, i_k \in N\}$$

R can be represented as a union or *disjunction* of IP/LP problems over the $\mathsf{S}(i_j)$. The number of different IP/LP problems that are necessary gives us the number of extensions of the rule for $\mathsf{S}(i)$ and f.

We arrive at a new complexity classification of many–valued logics. Given a logic with many–valued connectives and a certain set of signs our complexity measure will be the maximal number of disjuncts needed in the representation of $R_f(\mathsf{S}(i))$ for any connective f and sign $\mathsf{S}(i)$.

From this point of view the complexity of most many–valued logics in the literature, including Łukasiewicz logic, Post logic and classical logic is essentially the same, namely 2. Note that this complexity measure reflects *topological* properties of the truth tables, since the minimal number of extensions of a rule corresponds to the minimal number of convex regions needed to cover the relevant entries in the truth table.

We can look at the technique as a method to extract the classical part from many–valued logics. The size and shape of proof trees for classical and many–valued formulas constructed with connectives of the same complexity type is the same, while the many–valued instances have IP/LP problems attached to them.

Mixed Integer Programming and Disjunctive Methods

We will now see that constraint tableaux can in fact be linearized, in other words, a tableau can be translated into a single constraint system. In Operations Research merging of disjunctively connected IP/LP problems has been extensively investigated. Given a set of IP/LP problems over the same set of variables the task is to find a single constraint system,

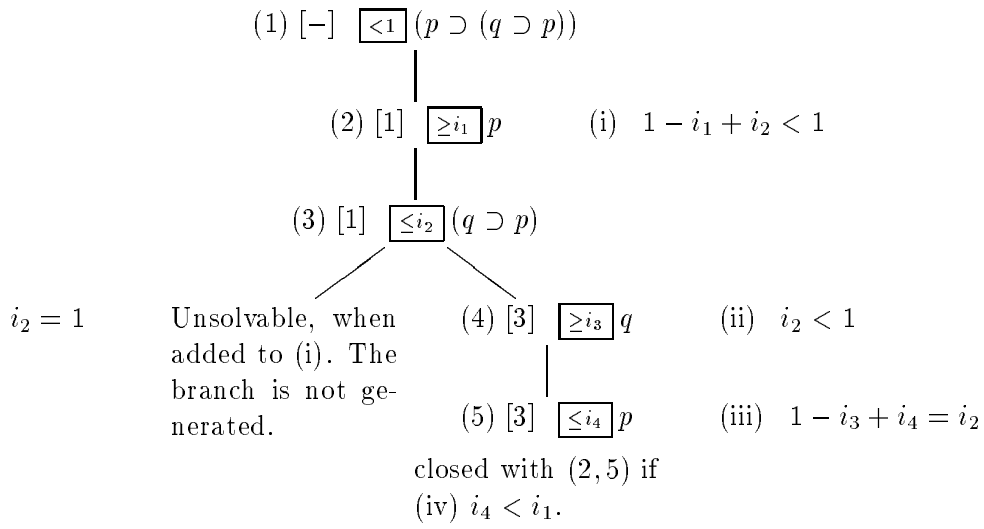

Figure 3: A Derivation in Infinitely Valued Łukasiewicz Logic using Constraints.

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

Table 3: Classical Tableau Rules in Disjunctive Constraint Formulation.

possibly involving both integer and rational variables, such that it comprises exactly the disjunction of the solutions of the input systems. The discipline is called *Disjunctive Programming* and we will refer to a constraint system involving linear inequalities over real *and* integer variables with the term *Mixed Integer Programming* (MIP) problem. Fairly recent overviews on MIP research can be found in [7, 8].[5]

Our first example will be a formulation of tableau rules for classical propositional logic which consists only of *linear rules*. It is given in Table 3. Double negations are always eliminated before rule application. Note that only two rules introduce new sign variables at all and thus have associated constraints. The MIP problems resulting from tableaux constructed with these rules are in fact pure IP problems as is the case for all finitely–valued logics.

A prototype implementation of a model checker for classical propositional logic has been implemented with the constraint mechanism of PrologIII, a logic programming lan-

[5] In the former work some connections to automated deduction in classical logics are explained. Recently, some very fast satisfiability checking algorithms for classical propositional logic have been designed on the basis of LP, see [9]. The difference between these approaches and the present one, besides that the former are only suitable for classical logic, is that they are motivated by resolution or the Davis–Putnam procedure while our approach is motivated by the tableau method. One consequence is that we are not confined to use conjunctive normal form for the input.

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

Table 4: Improved Constraint Rules for Classical Logic.

guage which is able to handle linear inequality constraints. First results are encouraging, for instance, the pigeon hole problem for $n = 8$ is solved in few minutes. The logic capabilities of PrologIII are hardly needed, after the input is transformed into an IP problem; we took PrologIII only for the benefit of having a prototype without having to spend much time on coding. We expect that the performance can be increased considerably by experimenting with the IP representation and using a tailor–made IP solving algorithm implemented in C.

Many other representations of tableau rules than the one in Table 3 are possible and some of them pay in increased efficiency in an implementation. A variant of the two more complicated of the classical constraint rules where one sign variable is saved is shown in Table 4.[6] The prize is to admit linear expressions as signs, but this is a mere technical difficulty which does not cause any problems. A similar optimization is possible for the rules of Łukasiewicz logic. Moreover, it can be shown that the standard translation of propositional CNF formulas into IP problems [9] can be obtained as a special case of our approach.[7]

Negation

As in conventional tableaux there are two possible ways of dealing with negation. The first one was chosen in the rules above; negation operators in front of complex connectives are treated together with these connectives as α– or β–formulas, double negations are eliminated and negated atoms are transformed into inequalities. This works only for some many–valued logics. The second method consists of viewing negations as proper connectives with their own rules and this works for all kinds of many–valued logics. Possible rules for negation (whenever it is defined as $\neg i = 1 - i$) are

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

MIP Formulation of Infinitely Valued Logic

Our next example of disjunctive methods will be a linear variant of Łukasiewicz infinitely valued logic. The only non–linear rule from Table 2 was the one for $\boxed{\leq i}$. As a first step let us give a slightly more redundant formulation of that rule:

[6] The rules are even sound and complete when the constraints are omitted completely. However, although redundant, the information reduces time for solving the IP considerably.

[7] We will present the details in a forthcoming paper.

$$\begin{array}{lll} & \boxed{\leq i}\,(\phi_1 \supset_{\text{Ł}} \phi_2) & \\ \hline & \boxed{\geq i_1}\,\phi_1 & i \leq 1 \\ i = 1 & \boxed{\leq i_2}\,\phi_2 & 1 - i_1 + i_2 = i \end{array}$$

The only difference is $i \leq 1$ instead of $i < 1$ in the right hand side constraint. A moment's thought reveals that this rule is still sound; when $i = 1$ some of the truth table entries in $\boxed{\leq 1}$ are now covered in both branches. This is not dramatic, however, since it happens also in the usual classical β–rules. Applications of disjunctive representation methods [7] and simplification then yields a linear formulation of the same rule

$$\begin{array}{ll} \boxed{\leq i}\,(\phi_1 \supset_{\text{Ł}} \phi_2) & \\ \hline \boxed{\geq i_1}\,\phi_1 & y \leq i \leq 1, \quad i_1 \leq 1 - y \\ \boxed{\leq i_2}\,\phi_2 & 1 - i_1 + i_2 = y + i, \quad y \leq i_2 \end{array}$$

where y is binary and i, i_1, i_2 range over $[0, 1]$. If $y = 0$ the right extension of the rule above is selected, the left extension if $y = 1$. For this reason y is called *control variable*. Hence, we can transform every many–valued deduction problem from Ł_ω into a single MIP problem whose integer part has not more variables than the input formula has connectives.

Corollary 6 $SAT_{\text{Ł}_\omega} \in NP$.

This result was obtained in [10] in a rather more complicated way using McNaughton's Theorem. Our completely different method is not only much simpler, but renders itself also to many other logics for which such results as McNaughton's Theorem do not exist.[8]

It is not trivial to compute MIP representations of many–valued tableau rules as can be seen in the example above, but the work done by Jeroslow and other researchers in the field of Operations Research, where a considerable amount of knowledge about MIP methods has been accumulated, fits in here exactly.

Further Research

Our presentation remains on a somewhat sketchy level and many details have to be filled in yet. Nevertheless we hope to have convinced the reader that there are natural and promising connections between classical tableaux and IP on the one side and between many–valued tableaux and MIP on the other. We summarize the directions for further research which are in our eyes the most promising:

[8] The other direction, NP–hardness, was shown, too, in [10]. The idea is to define for each set of propositional variables $p_1, \ldots p_k$ a Ł_ω–formula $\texttt{two}(p_1, \ldots p_k)$ such that $\texttt{two}(p_1, \ldots p_k)$ is satisfiable in Ł_ω iff $p_1, \ldots p_k$ are assigned binary truth values. Then for any formula ϕ which contains the propositional variables $p_1, \ldots p_k$ it is true that ϕ is satisfiable in classical logic iff $\texttt{two}(p_1, \ldots p_k) \supset \phi$ is satisfiable in Ł_ω. If `two` has polynomial size in k this property reduces SAT to $\text{SAT}_{\text{Ł}_\omega}$. In Ł_ω the function `two` is a bit awkward to define, since there is no connective corresponding to truth value set complement. The technique works for many other non–standard logics.

- Apply fast satisfiablity checkers for many–valued logics to fuzzy reasoning and verification of integrated circuits.
- The translation of tableau speed–up methods such as additional inference rules, lemma generation, indexing etc. into the MIP representation should be investigated.
- While going from search trees to MIP representations structural information is lost. On the other hand, we might propagate such information from the tableau to the MIP representation, for instance, by specifying a partial order on variables which is then used to determine the order in which they are fixed when solving the MIP.
- IP is hard to do while LP is not. It has been shown in other contexts that with certain inference rules (for example unit resolution) it is safe to substitute LP for IP. It would be interesting to identify such situations in the present context.
- Perhaps the most interesting and challenging task is the extension to first–order logic. Several approaches are possible and are currently investigated.
- The technique may well be applicable to other nonclassical logics, such as modal or temporal logics.

Acknowledgements

I am grateful to Daniele Mundici and Klaus Ries for valuable discussions and suggestions.

References

[1] R. Smullyan. *First–Order Logic*. Springer, New York, second edition, 1968.

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

[3] A. Urquhart. Many–valued logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. III: Alternatives in Classical Logic*, chapter 2, pages 71–116. Reidel, Dordrecht, 1986.

[4] R. Hähnle. *Tableaux–Based Theorem Proving in Multiple–Valued Logics*. PhD thesis, University of Karlsruhe, Dept. of Computer Science, May 1992.

[5] 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.

[6] 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.

[7] R. S. Jeroslow. *Logic–Based Decision Support. Mixed Integer Model Formulation*. Elsevier, Amsterdam, 1988.

[8] G. L. Nemhauser and L. A. Wolsey. Integer programing. In G. L. Nemhauser, A. H. G. Rinnooy Kan, and M .J. Todd, editors, *Handbooks in Operations Research and Management Science, Vol. II: Optimization*, chapter VI, pages 447 – 527. North–Holland, Amsterdam, 1989.

[9] J. N. Hooker. Logical inference and polyhedral projection. In *Proceedings Computer Science Logic Workshop 1991, Berne*. Springer LNCS, 1991.

[10] D. Mundici. Satisfiability in many-valued sentential logic is NP–complete. *Theoretical Computer Science*, 52:145 – 153, 1987.