Original paper

Soft Computing 01 (1997) 107–119 © Springer-Verlag 1997

Proof theory of many-valued logic–linear optimization–logic design: connections and interactions

R. Hähnle

Abstract In this paper proof theory of many-valued logic is connected with areas outside of logic, namely, linear optimization and computer aided logic design. By stating these not widely-known connections explicitly, I want to encourage interaction between the mentioned disciplines. Once familiar with the others' terminology, I believe that the respective communities can greatly benefit from each other.

Key words many-valued logic, mixed integer programming, logic design

1
Introduction

The intention of this paper is not mainly to prove new results, rather, known results in the proof theory[1] of many-valued logic are connected with other areas such as linear optimization and computer aided design of logic circuits (*logic design* for short).

These connections provide the possibility for fruitful interaction between the mentioned disciplines, however, they are not widely known among researchers in either area. Once familiar with the others' terminology, I believe that the respective communities can greatly benefit from each other.

One reason for the lack of interaction is that while some relationships are well-known and straightforward in the two-valued case, the connection becomes a lot deeper and interesting in the many-valued case. Another is the familiar phenomenon of different communities talking different languages.

Hence, one purpose of this paper is to provide the core of a concordance, in this case between proof theory and logic design. The second goal is to point out explicitly several spots, where proof theory and linear optimization, respectively, proof theory and logic design could benefit from each other.

I tried to write this paper in such a style that it is accessible for a wide audience and I made an effort to render it self-contained.

In Section 2 definitions and basic results of many-valued proof theory are summarized. In Section 3 connections between many-valued logics and mixed integer programming are explored. In Section 4 many-valued logic proof theory is stretched into a quite different area: there, I give a partial mapping between the vocabularies of proof theory and automated theorem proving on the one side and BDDs and logic design on the other.

Sections 3 and 4 can be read quite independently, but both require familiarity with Section 2.

2
The basics

All I want to tell in this paper can be told at the propositional level. Thus I deal only with propositional many-valued logics.

2.1
Many-valued logic: syntax

Definition 1 Let Σ be a propositional signature, that is, a denumerable set of propositional variables $\{p_0, p_1, \dots\}$. Σ is also called the set of atomic formulas (or atoms for short). □

Definition 2 A propositional language is a pair $\mathbf{L} = \langle \Theta, \alpha \rangle$, where Θ is a finite or denumerable set of logical connectives and $\alpha : \Theta \to \mathbb{N}$ defines the arity of each connective. Connectives with arity 0 are called logical constants.

The set L_Σ of $\mathbf{L}$-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 $\phi_1, \dots, \phi_m \in L_\Sigma$, $\theta \in \Theta$ and $\alpha(\theta) = m$ then $\theta(\phi_1, \dots, \phi_m) \in L_\Sigma$. □

Notation A propositional language $\langle \Theta, \alpha \rangle$ with a finite set of connectives $\Theta = \{\theta_1, \dots, \theta_r\}$ is denoted $\langle \theta_1/\alpha(\theta_1), \dots, \theta_r/\alpha(\theta_r) \rangle$. Moreover, the usual conventions on bracketing for well-known connective symbols (which are assumed to have the usual precedence order) are being made. For binary connectives, infix notation is used.

Received: *20 March 1997*/ *Accepted*: *1 April 1997*

R. Hähnle
Institute for Logic, Complexity and Deduction Systems
Department of Computer Science, University of Karlsruhe,
D-76128 Karlsruhe, Germany
reiner@ira.uka.de
http://i12www.ira.uka.de/ ~ reiner

This paper is based on a talk given in 1996 at the ESSLLI Symposium on *Proof Theory and Computational Aspects of Many-Valued Logics* in Prague. A preliminary version appeared in the third volume of the Annals of the Kurt-Gödel-Society, Springer-Verlag, Wien

I would like to thank Dr. Matthias Baaz for inviting me to the ESSLLI 1996 Symposium on *Proof Theory and Computational Aspects of Many-Valued Logics*.

[1] In this article I always mean proof theory in the tradition of Gentzen, not of Frege or Hilbert

Example 1 Following Mundici [29], the language of Łukasiewicz logic is given by $\mathbf{L}_{\mathrm{Luk}} = \langle \neg/1, \oplus/2, \odot/2 \rangle$. Examples of $\mathbf{L}_{\mathrm{Luk}}$-formulas are: $(p \odot q) \oplus ((p \oplus q) \odot r)$, $(\neg p \odot \neg q) \oplus ((\neg p \oplus \neg q) \odot \neg r)$. □

2.2
Many-valued logic: semantics

Definition 3 The set of truth values N is either the unit interval on the rational numbers, denoted with $[0, 1]$, or it is a finite set of rational numbers of the form $\{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\}$, where $n \in \mathbb{N}$. In either case $|N|$ denotes the cardinality of N, in particular, $|[0, 1]| = \aleph_0$. □

It is stressed that fractional numbers are used to represent truth values merely, because they are convenient to define several well-known logics. If nothing else is said, we do not make any use of their special properties, in particular, truth values are assumed to be unordered if not stated otherwise. Let $\mathscr{P}^+(N)$ denote the non-empty subsets of a truth value set N.

Definition 4 Let ϕ be a formula and $S \in \mathscr{P}^+(N)$. Then we call the expression $S\ \phi$ signed formula. If p is an atomic formula, then $S\ p$ is a signed literal. A signed formula that is not a literal is a complex signed formula. □

Informally, a signed formula states that it takes on truth values contained in its sign, see Def. 9 below.

Definition 5 If $\mathbf{L} = \langle \Theta, \alpha \rangle$ is a propositional language then we call a pair $\mathbf{A} = \langle N, A \rangle$, where N is a set of truth values and A assigns to each $\theta \in \Theta$ a function[2] $A(\theta) : N^{\alpha(\theta)} \to N$ a (propositional) matrix for $\mathbf{L}$. The range $\mathrm{rg}(\theta)$ of a connective θ is defined as the range of the function $A(\theta)$:

$$\mathrm{rg}(\theta) = \{A(\theta)(i_1, \ldots, i_{\alpha(\theta)}) \mid i_1, \ldots, i_{\alpha(\theta)} \in N\}$$ □

Definition 6 A pair $\mathscr{L} = \langle \mathbf{L}, \mathbf{A} \rangle$ consisting of a propositional language and a matrix for it is called many-valued or $|N|$-valued propositional logic. □

If no confusion can arise we use the same symbol for θ and $A(\theta)$. Moreover, we note that with each formula ϕ of a many-valued logic containing variables $\{p_1, \ldots, p_k\}$ a function $f_\varphi : N^k \to N$ is associated in a natural way: f_{p_i} is the projection to the i-th component and $f_{\theta(\varphi_1, \ldots, \varphi_{\alpha(\theta)})} = A(\theta)(f_{\varphi_1}, \ldots, f_{\varphi_{\alpha(\theta)}})$ otherwise (cf. Definition 7 below). Again, we normally use the same symbol for ϕ and f_φ.

Sometimes a logic is equipped with a non-empty subset D of the set of truth values called the designated truth values which play the rôle of the truth values which are considered to affirm satisfability.

Example 2 Let N be arbitrary and $n = |N|$. Then we define the family of n-valued Łukasiewicz logics to be the propositional logics with language $\mathbf{L}_{\mathrm{Luk}}$, designated truth values $D = \{1\}$, and the matrix given by:

$$\neg i = 1 - i \tag{1}$$

$$i \oplus j = \min\{1, i + j\} \tag{2}$$

$$i \odot j = \max\{1, i + j - 1\} \tag{3}$$

The family of n-valued Kleene logics relative to the language $\mathbf{L}_{\mathrm{Kle}}$ is defined by

$$\neg i = 1 - i \tag{4}$$

$$i \vee j = \max\{i, j\} \tag{5}$$

$$i \wedge j = \min\{i, j\} \tag{6}$$

The family of n-valued Gödel logics relative to the language $\mathbf{L}_G$ is defined by

$$\neg_G i = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

$$i \to_G j = \begin{cases} 1 & \text{if } i \le j \\ j & \text{otherwise} \end{cases} \tag{8}$$

In each case min, max, $+$, $-$, $\le$ are interpreted wrt the natural order on N. In Figs. 1, 2, and 4, respectively, the function graphs of Łukasiewicz sum and Kleene disjunction in two-valued, three-valued, and infinite-valued logic, are shown. In the two-valued case (Fig. 1) both graphs are identical and reduce to classical disjunction. In the three-valued case (Fig. 2) they differ only in the highlighted function value while only from the infinite-valued connectives (Fig. 4) their completely different nature is obvious.

In Fig. 6 function graphs of the connectives of infinite-valued Gödel logic are displayed. □

Definition 7 Let $\mathscr{L}$ be a propositional logic. A (propositional) (Σ-)interpretation is a function $\mathbf{I} : \Sigma \to N$. $\mathbf{I}$ is extended to arbitrary $\phi \in L_\Sigma$ in the usual way:

1. If ϕ is a logical constant then $\mathbf{I}(\phi) = A(\phi)$.
2. If $\phi = \theta(\phi_1, \ldots, \phi_r)$, then $\mathbf{I}(\theta(\phi_1, \ldots, \phi_r)) = A(\theta)(\mathbf{I}(\phi_1), \ldots, \mathbf{I}(\phi_r))$. □

Definition 8 Let $S \subseteq N$. A formula ϕ is said to be S-satisfiable iff there is an interpretation I such that $\mathbf{I}(\phi) \in S$. We say then that $\mathbf{I}$ is an S-model of ϕ. ϕ is an S-tautology, in symbols $\models \phi$, iff every interpretation S-satisfies ϕ. □

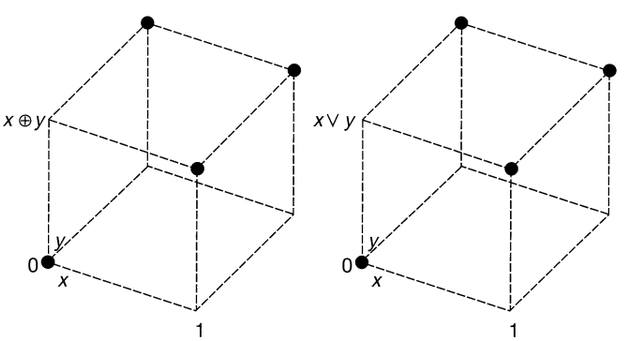

Fig. 1. Function graphs of x $\oplus$ y and $x \vee y$ on $N = \{0, 1\}$

[2] If $\alpha(\theta) = 0$ then the usual convention $A(\theta) \in N$ is made

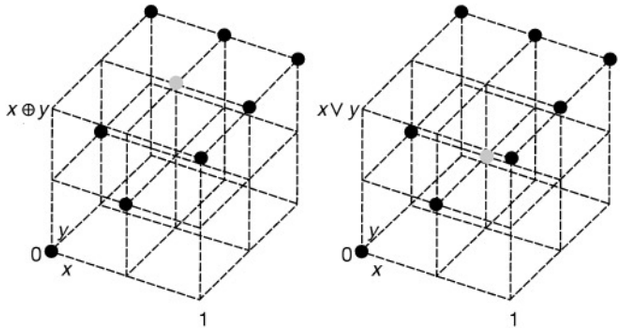

Fig. 2. Function graphs of $x \oplus y$ and $x \vee y$ on $N = \{0, \frac{1}{2}, 1\}$

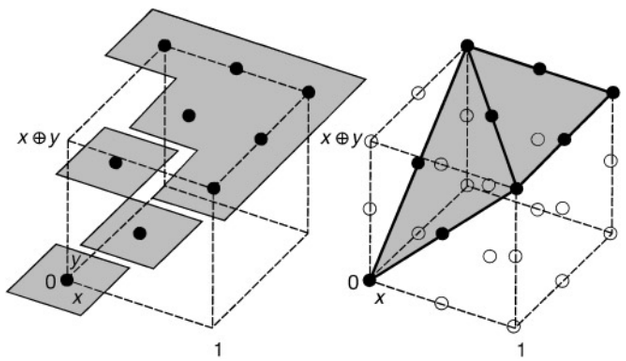

Fig. 3. Two ways of representing the graph of $x \oplus y$ over $\{0, \frac{1}{2}, 1\}$

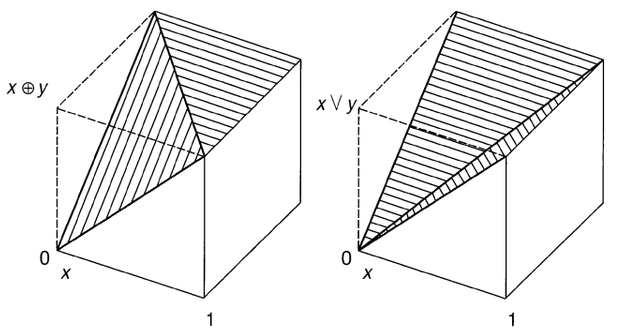

Fig. 4. Function graphs of $x \oplus y$ and $x \vee y$ on $N = [0, 1]$

Definition 9 A signed formula $S\,\phi$ is satisfiable iff $\mathbf{I}(\phi) \in S$ for some $\mathbf{I}$. □

Definition 10 Let $S \subseteq N$. A formula ψ is a logical S-consequence of ϕ, in symbols $\phi \models \psi$ iff every S-model of ϕ is an S-model of ψ. ϕ and ψ are logically S-equivalent iff each is a logical S-consequence of the other. □

If a logic is equipped with designated truth values D, and if D is obvious from the context, then we say satisfiable instead of D-satisfiable, model instead of D-model etc. If $N = \{0, 1\}$ and $D = \{1\}$ then, of course, all notions collapse to the standard ones of classical logic.

2.3
Representation of finite-valued connectives

In the present subsection we state and prove a fundamental result regarding the representation of finite-valued connectives or, equivalently, of functions over finite domains. The result is straightforward, but nevertheless it constitutes the theoretical basis of tableaux and sequent calculi for finite-valued propositional logics.

As is the case for other non-classical logics, proof calculi for deduction in MVL can be roughly divided into two classes:

Internal calculi: the objects constructed during a proof are expressed in the same language as the goal to be proven; a typical example are Hilbert type calculi.

External calculi: formal proofs are over an *extended* language that may involve elements from the semantics such as designators for truth values, worlds or even non-logical expressions such as constraints; a typical example are signed semantic tableaux.

Purists among proof theorists often only accept calculi of the first kind and regard the second option as "cheating". On the other hand, if a uniform and computationally efficient treatment of deduction is desired, there seems to be no alternative to external calculi.

A somewhat extreme position of gaining a classical logic approach to deduction in non-classical logic would be to formulate the "external" elements in the second approach as a meta theory in classical logic. For a wide range of logics this is even possible within first-order logic. The "meta theory" of finite-valued logic in particular can always be captured without having to move to a higher-order stage.[3] On the other hand, classical *propositional* logic is too weak for an adequate representation of most many-valued semantics.

It turns out that in the case of infinite-valued logic even signed formulas are not enough which leads to incorporation of linear arithmetic constraints and, ultimately, to a reduction of infinite-valued deduction to mixed integer programming. Later we will see that mixed integer programming is in a certain sense an adequate formalism to handle infinite-valued logic.

We start now with the finite-valued case.

Theorem 1 *Let N be finite, $n = |N|$, $\mathbf{S} \subseteq \mathscr{P}^+(N)$ a family of truth values satisfying*

$$\text{For all } i \in N \text{ there are } S_1, \ldots, S_k \in \mathbf{S} \text{ such that } \bigcap_{j=1}^{k} S_j = \{i\}. \tag{9}$$

If $\phi = S\,\theta(\phi_1, \ldots, \phi_m)$ $(m \geq 1, S \in \mathbf{S})$ is a signed formula from an n-valued logic $\mathscr{L} = \langle \mathbf{L}, \mathbf{A} \rangle$ such that $S \cap rg(\theta) \neq \emptyset$, then there are numbers $M_1, M_2 \leq n^m$, index sets $I_1, \ldots, I_{M_1}$, $J_1, \ldots, J_{M_2} \subseteq \{1, \ldots, m\}$, and signs $S_{rs}, S_{kl} \in \mathbf{S}$ with $1 \leq r \leq M_1$,

[3] This is not necessarily true for infinite-valued logic, see [39]

$1 \leq k \leq M_2$ *and* $s \in I_r$, $l \in J_k$ *such that*

$$\phi \text{ is satisfiable iff } \bigvee_{r=1}^{M_1} \bigwedge_{s \in I_r} (S_{rs}\ \phi_s \text{ is satisfiable})$$

$$\text{iff } \bigwedge_{k=1}^{M_2} \bigvee_{l \in J_k} (S_{kl}\ \phi_l \text{ is satisfiable}),$$

where $\bigvee$ *and* $\bigwedge$ *denote classical meta disjunction and conjunction, respectively, and have their usual meaning. We call the first expression a sets-as-signs DNF representation of* ϕ*, the second a sets-as-signs CNF representation of* ϕ*.*

This theorem is proved in many places for the case $S \in \mathbf{S} = \{\{i\} | i \in N\}$, for instance, in [34, 46, 13, 48, 2]. The general case was first handled in [16], see also [17].

Observe that for guaranteeing the existence of DNF/CNF representations "enough" signs must be available: **S** has to satisfy (9). A linear number of signs is sufficient as $\mathbf{S} = \{\{i\} | i \in N\}$ clearly satisfies (9), but the choice of **S** has a considerable impact on the size (i.e. the numbers M_1 and M_2) of the resulting representations. This topic is further discussed in [17].

Example 3 Below are DNF, resp., CNF representations of a signed formula in three-valued Łukasiewicz logic. The representations are denoted like rules to enhance readability. CNF representations use double vertical bars to distinguish them from DNF representations.

$$\frac{\{0, \tfrac{1}{2}\}\ \phi \oplus \psi}{\begin{array}{l|l} \{0, \tfrac{1}{2}\}\ \phi & \{0\}\ \phi \\ \{0\}\ \psi & \{0, \tfrac{1}{2}\}\ \psi \end{array}} \qquad \frac{\{0, \tfrac{1}{2}\}\ \phi \oplus \psi}{\begin{array}{l\|l\|l} \{0, \tfrac{1}{2}\}\ \phi & \{0\}\ \phi & \\ & \{0\}\ \psi & \{0, \tfrac{1}{2}\}\ \psi \end{array}} \tag{10}$$

□

2.4 Many-valued deduction based on clause form representations

The significance of Theorem 1 is that it directly gives rise to rules in generic sequent and tableau calculi for finite-valued logics. In fact we identify DNF representations with those rules:

Definition 11 If $\bigvee_{r=1}^{M} \bigwedge_{s \in I_r} S_{rs}\ \phi_s$ is a sets-as-signs DNF representation of $\phi = S\ \theta(\phi_1, \ldots, \phi_m)$ $(m \geq 1)$, then a many-valued sets-as-signs tableau rule for ϕ is defined as

$$\frac{S\ \theta(\phi_1, \ldots, \phi_m)}{C_1 | \cdots | C_M},$$

where $C_r = \{S\ \phi_s | s \in I_r\}$. The C_r are called extensions of the rule. □

The notion of a tableau for a signed formula $S\ \phi$ is defined exactly as in the classical case [43, 15], but with respect to many-valued sets-as-signs rules.

Definition 12 One defines a many-valued sets-as-signs tableau for a finite set of signed formulas Φ as a directed tree labeled with signed formulas and constructed as follows:

1. A linear tree whose labels are exactly the formulas in Φ is a many-valued sets-as-signs tableau for Φ.
2. If **T** is a many-valued sets-as-signs tableau for Φ, ϕ a complex signed formula occurring on a branch B of **T**, and $C_1, \ldots, C_M$ the extensions of a sets-as-signs tableau rule for ϕ, then a many-valued sets-as-signs tableau for Φ is obtained from **T** by extending B with $|M|$ many new linear branches, where the labels of the r-th branch are exactly the members of C_r.
3. No other tree is a many-valued sets-as-signs tableau for Φ.

Some care must be spent on the definition of closure:

Definition 13 Let **T** be a many-valued sets-as-signs tableau and B one of its branches. B is closed iff one of the following conditions holds:

1. There are signed formulas $S_1\ \phi, \ldots, S_r\ \phi$ on B such that $\bigcap_{i=1}^{r} S_i = \emptyset$. In this case we say that (the set of signed formulas on) B is inconsistent.[4]
2. There is a signed formula $S\ \theta(\phi_1, \ldots, \phi_m)$ $(m \geq 0)$ on B such that $S \cap \mathrm{rg}(\theta) = \emptyset$.

T is closed iff each of its branches is closed. A branch that is not closed is called open. □

Theorem 2 ([16]) *Let* ϕ *be a formula in a finite-valued logic and let* $\emptyset \neq S \subseteq N$*. Then* ϕ *is S-valid iff there exists a finite, closed many-valued sets-as-signs tableau for* $(N \backslash S)\ \phi$*.*

As in classical logic it is sufficient to apply each rule at most once to each complex formula on a branch. Therefore, many-valued sets-as-signs tableaux can assumed to be finite.

We saw that recursive application of DNF representations to all complex subformulas of a signed formula yields a tableau procedure. The usual semantics of a tableau is that the disjunction over the conjunction of the formulas in its branches is satisfiable, thus corresponding to a classical DNF over the signed formulas occurring on its branches. Hence, a DNF over signed literals can be computed by identifying its (conjunctive) clauses with the signed literals occurring on an open branch of a fully expanded tableau.

Just in the same way a CNF over signed literals can be computed for a given signed formula using CNF representations instead of DNF representations. Once a CNF over signed literals has been obtained, generalized versions of resolution [2, 19], of the Davis-Putnam-Loveland procedure [20], etc. can be employed for a deductive treatment of many-valued logic.

One problem of DNF and CNF representations is that their size (the numbers M_1 and M_2 in Theorem 1) depends on the cardinality of the truth value set of the underlying logic. If

[4] Just as in classical logic it is sufficient to restrict the definition to the case when ϕ is atomic. Accordingly one says that B is atomically inconsistent

a logic has more than a few truth values, then the resulting tableaux and CNFs become intolerably large in the worst case. The bad news is that the worst case is in fact achieved for some well-known and important connectives, for example, Łukasiewicz sum (2) and product (3).

In classical logic the growth at least of CNFs can be kept within polynomial bounds with so-called *structure preserving CNF transformations* [47, 32]. The basic idea is to use a polynomial size transformation of a complex formula ϕ into a satisfiability equivalent formula $\bigwedge_i \phi_i$ such that each ϕ_i is of constantly bounded depth. As a consequence, the CNF of $\bigwedge_i \phi_i$ and, therefore, of ϕ, has polynomial size. Such a transformation can be easily described as:

For each proper complex subformula ψ of ϕ:

replace ϕ with $\phi\{p_\psi/\psi\} \wedge (p_\psi \leftrightarrow \psi)$, where p_ψ is a new atom.[5] (11)

The result is a formula of the form $p_\varphi \wedge \bigwedge_\psi (p_\psi \leftrightarrow \psi)$. Thus each complex subformula ψ is replaced by its *abbreviation* or *definition* p_ψ. Each equivalence of the form $p_\psi \leftrightarrow \psi$ says that p_ψ is equivalent to the subformula it abbreviates. The CNF of each equivalence is of constant length: all proper complex subformulas in ψ were themselves abbreviated, hence all the ψ contain exactly one connective. Finally, as the number of complex subformulas is linear (in the size of ϕ), the CNF of $p_\varphi \wedge \bigwedge_\psi (p_\psi \leftrightarrow \psi)$ is polynomial in the size of ϕ. Exactly the same technique can be used in the many-valued case (let ϕ be a signed formula):

For each unsigned proper complex subformula ψ of ϕ:

replace ϕ with $\phi\{p_\psi/\psi\} \wedge \{1\}\ (p_\psi \Leftrightarrow \psi)$, where p_ψ is a new atom. (12)

Here, $\Leftrightarrow$ is a connective playing a similar rôle as classical equivalence and which is definable in Łukasiewicz logic (and, similarly, in many other logics) as $p \Leftrightarrow q = (\neg p \oplus q) \wedge (p \oplus \neg q)$ or, explicitly, as:

$$i \Leftrightarrow j = \min\{1, 1-i+j, 1+i-j\} \tag{13}$$

3 Satisfiability in many-valued logic as mixed integer programming

3.1 Limits of clause form representations

Even if one uses a polynomial size CNF transformation Theorem 1 still cannot be used as a basis for deduction in the infinite-valued case. Although it is possible to generalize it to an infinite number of truth values, the resulting representations become infinite as well, thus leading to infinitely branching tableau, resp., translation rules already for formulas with only one connective.

The reason for these difficulties is that so far we have been using classical propositional logic plus signs as a meta language to characterize many-valued connectives. This is very desirable from a deductive point of view, because we can make use of all the deductive machinery available in classical logic, but the approach is limited: classical propositional logic is simply too poor to yield straightforward finite characterizations of infinite-valued logics (later we will see better why this is so).

The extension of classical logic I have in mind to solve the current problems is best motivated with an example. Let us have a look at the graph of Łukasiewicz sum with three truth values depicted as the solid spheres in the left part of Fig. 2. A signed formula of the form $S\ \phi \oplus \psi$ defines a subset of this graph. A CNF/DNF representation of the signed formula uses intersections/unions of rectangular areas to build the required subset, schematically displayed in the left part of Fig. 3.

The DNF/CNF representation obviously is not independent of the cardinality of the truth value set. In particular, when using this technique for infinite-valued functions an infinite number of rectangles is required in the worst case.

Looking at the graph of $\oplus$ over $N = [0, 1]$ (left part of Fig. 4), on the other hand, suggests a quite different way of representation which is much less dependent on the number of truth values: to represent $\oplus$ over, say, $N' = \{0, \frac{1}{2}, 1\}$ simply intersect the graph of $\oplus$ over $N = [0, 1]$ with $N' \times N'$. This is illustrated in the right part of Fig. 3, where the spheres represent $N' \times N'$, the solid ones being as well in the graph of $\oplus$.

Thus we arrive at the question of how the graph of $\oplus$ or, more generally, of how the graph corresponding to an arbitrary many-valued formula can be represented.

3.2 McNaughton's Theorem

It turns out that for Łukasiewicz logic there is a classical solution to this problem which gives a clue on how to proceed for other logics as well.

Definition 14 A function $f: [0, 1]^h \to [0, 1]$ is a McNaughton function if

1. f is continuous wrt the natural topology of $[0, 1]^k$ and
2. it is piecewise linear with integral coefficients i.e. there is a finite number of linear polynomials $p_i(x_1, \ldots, x_k)$ with integral coefficients such that for each $\vec{x} \in [0, 1]^k$ there is an i with $p_i(\vec{x}) = f(\vec{x})$. □

Theorem 3 ([27]) 1. *If $\phi \in \mathbf{L}_{\mathrm{Luk}}$ then f_φ is a McNaughton function.*

2. *Let $f: [0, 1]^k \to [0, 1]$ be a McNaughton function. Then there is a formula $\phi \in \mathbf{L}_{\mathrm{Luk}}$ such that $f = f_\varphi$.*

The second part of the theorem is the hard one. It gives a deep characterization of infinite-valued Łukasiewicz logic. The first part of the theorem is easy to prove, for instance, by noting that $f_{\neg x}$, $f_{x \oplus y}$, and $f_{x \odot y}$ are McNaughton functions by definition, cf. (1–3). A straightforward induction yields the result. The number of polynomials required for f in the second part of the theorem is in general exponential in the number of connectives of ϕ.

[5] $p\{q/r\}$ stands for simultaneous replacement of each occurrence of r in p with q

3.3
Mixed integer programming

In this subsection some bare facts and definitions about Mixed Integer Programming (MIP) are given. As a background reading, for example, [40, 25] is recommended.

It is a well known fact (see, for example, [22, 25]) that propositional classical CNF formulas correspond to certain 0–1 integer programs. More precisely, given a set Φ of classical clauses over the signature Σ one transforms each clause

$$p_1 \vee \cdots \vee p_k \vee \neg p_{k+1} \vee \cdots \vee \neg p_{k+m} \tag{14}$$

into a linear inequation

$$\sum_{i=1}^{k} p_i - \sum_{j=k+1}^{m} p_j \geqslant 1-m \tag{15}$$

Here, the variables from Σ are interpreted as *polynomial variables* ranging over $\{0, 1\}$. It is easy to see that the resulting set of inequations is solvable iff Φ is satisfiable.

With the expression linear inequation we mean in the following always a term of the form $a_1p_1 + \cdots + a_mp_m \geq c$, where $a_1p_1 + \cdots + a_mp_m$ is a linear polynomial over $\{p_1, \ldots, p_m\}$ with integral coefficients, where the variables p_i either run over $\{0, 1\}$ or over N(N is as in Definition 3), and c is an integer. $a_1p_1 + \cdots + a_mp_m$ is called linear term.

Definition 15 Let $\mathbf{J}$ be a finite set of linear inequations and K a linear term. Let Σ be the set of variables occurring in $\mathbf{J}$ and K. Assume N is finite. Then $\langle \mathbf{J}, K\rangle$ is a (bounded) integer program (IP) with cost function K.[6] If N is infinite, then we have a (bounded) 0–1 mixed integer program (MIP). When all variables in Σ run over infinite N we have a (bounded) linear program (LP).

A variable assignment $\sigma : \Sigma \to \{0, 1\} \cup N$ that respects the type of each variable and such that all inequations in $\mathbf{J}\sigma$ are satisfied in called a feasible solution of $\langle \mathbf{J}, K\rangle$. A variable assignment to σ such that the value of K is minimal among all feasible solutions is called an optimal solution. $\langle \mathbf{J}, K\rangle$ is feasible iff there are feasible solutions. □

In the following, when I speak of (M)IP/LPs, only the set of inequations $\mathbf{J}$ is meant. If there is a cost function as well, it will be explicitly mentioned.

Definition 16 Let $M \subseteq [0, 1]^k$. M is MIP-representable if there is an MIP $\mathbf{J}$ with variables $\Sigma' = \{x_1, \ldots, x_k\}$ over $[0, 1]$ and variables Σ'' over $\{0, 1\}$ such that

$$M = \{\vec{x} | \vec{x} \text{ is feasible solution of } \mathbf{J}\sigma \text{ for some } \sigma : \Sigma'' \to \{0, 1\}\}.$$

A many-valued logic is MIP-representable iff for all its connectives θ the graph of $A(\theta)$ is MIP-representable. The variable in an MIP-representation of a function graph that holds the function value is called output variable, the variables that hold the function arguments are called argument variables. □

All finite-valued logics are obviously MIP-representable.

Proposition 4 (see [40]) *Each of the problems to check whether a* 0–1 *MIP* (*resp., an IP*) *has feasible solutions and to find an optimal/feasible solution is NP-complete. The problem to check whether an LP has feasible solutions and to find an optimal/feasible solution is in P.*

3.4
McNaughton functions and MIP-representations

McNaughton's Theorem can be generalized in a way most important from the deductive point of view:

Theorem 5 (Generic MIP version of McNaughton's Theorem)

1. *If $\phi(\vec{p})$ is a formula of an MIP-representable logic then there is an MIP $\mathbf{J}_\varphi$ with argument variables $\vec{p}$ and output variable y whose feasible solutions restricted to $(\vec{p}, y)$ are the graph of $f_\varphi(\vec{p})$. Moreover, the size of $\mathbf{J}_\varphi$ is linear in the size of ϕ.*
2. *Let $\mathbf{J}$ be an MIP over variables Σ. Then there is a Σ-formula $\phi_\mathbf{J} \in \mathbf{L}_{Luk}$ which is satisfiable iff $\mathbf{J}$ is feasible.*

The first part of the theorem now is non-trivial to prove. In return, it provides a direct way to perform deduction in MIP-representable infinite-valued logics. The second part of the theorem simply says that there is a "backend" to McNaughton's result that allows to go from MIP to McNaughton functions. We will see that MIPs in fact are captured quite naturally by McNaughton functions.

We do not immediately start to prove Theorem 5, but apply it first to Łukasiewicz logic by showing its MIP-representability.

Proposition 6 *Infinite-valued Łukasiewicz logic is MIP-representable.*

Proof. An MIP-representation of the graph of $\oplus$ is given by the following MIP, where x and y are argument variables, i is output variable and z is an additional 0-1-variable.

$$\begin{aligned}
&(i)\ \ x+y+z-i \geq 0\\
&(ii)\ \ -x-y+z+i \geq 0\\
&(iii)\ \ x+y-z \geq 0\\
&(iv)\ \ -x-y+z \geq -1\\
&(v)\ -z+i \geq 0
\end{aligned} \tag{16}$$

To see this, first set $z=0$. Then the polynomial $p_1(x, y) = i = x+y$ is defined by (i, ii), inequations (iii, v) are trivially satisfied, and (iv) i.e. $x+y \leq 1$ determines the area in which p_1 equals $\oplus$. This is the part of the graph depicted on the left in Fig. 4 with vertical hatching.

Similarly, if $z=1$ the polynomial $p_2(x, y) = i = 1$ is given by (v), inequations (i, ii, v) are trivially satisfied, and (iii) i.e.

[6] The adjective *integer* is justified, because the elements of N can w.l.o.g. assumed to be of the form $\{0, 1, \ldots, n-1\}$

$x+y \geq 1$ determines the area in which p_2 equals $\oplus$. This is the part of the graph depicted on the left in Fig. 4 with horizontal hatching. An MIP-representation of the graph of $\neg$ is straightforward:

$$-x-i \geq -1$$

$$x+i \geq 1$$

Łukasiewicz product $\odot$ is handled by duality: $x \odot y = \neg(\neg x \oplus \neg y)$. □

In order to apply Theorem 5 to Łukasiewicz logic one needs to find MIP-representations of arbitrary formulas ϕ. This can be done by the following method which at the same time is a

Proof of theorem 5 part 1 Assume we have an MIP-representation of the graph of $A(\theta)$ with output variable y_φ and argument variables $x_{\varphi_1}, \ldots, x_{\varphi_k}$ for each complex subformula $\psi = \theta(\phi_1, \ldots, \phi_k)$ of ϕ (as provided by Proposition 6 in the case of Łukasiewicz logic).

Now connect the MIPs for each proper complex subformula ψ of ϕ by adding equations of the form $x_\psi = y_\psi$; furthermore, add equations $x_p = p$ for each propositional variable p of ϕ and obtain thus an MIP-representation of the graph of f_φ with output variable y_φ and argument variables p. The size of each MIP is constant and depends only on the connective θ and the number of all MIPs is proportional to the number of complex subformulas in ϕ, hence it is linear in the size of ϕ.

Here, the same technique as in structure preserving CNF transformations is used: by explicitly naming output variables they can be connected to many occurrences of corresponding argument variables. □

Given an MIP-representation of its graph, it is easy to check, say, $[c, d]$-satisfiability of a many-valued formula ϕ for given $0 \leq c \leq d \leq 1$: simply add the constraint $c \leq y_\varphi \leq d$, where y_φ is the output variable of ϕ and test the resulting MIP for feasibility. This is a notable improvement on the procedure given in [18], where cost functions were required.

From Theorem 5 part 1 and Proposition 4 immediately follows (and thus answering an open question raised in [18, p. 256]):

Corollary The $[c, d]$-satisfiability problem of any MIP-representable logic for $0 \leq c \leq d \leq 1$ is in NP.

A straightforward and concise implementation of a satisfiability checker for infinite-valued Łukasiewics logic based on these ideas is displayed in Fig. 5. It is written in Eclipse Prolog and can solve textbook examples within fractions of a second. To use it, issue a query such as

```
:- sat(I, plus(neg(atom(P)), atom(P))), I$<1.
```

The answer is "no" indicating that there is no interpretation such that $\neg p \oplus p$ evaluates to a truth value smaller than 1, in other words, it is a $\{1\}$-tautology.

The MIP-representation of the graph of $\oplus$ in (16) seemed to drop from the sky. How can MIP-representations be computed systematically?

```
:- lib(r).                  % load constraint solver

sat(I,plus(Phi,Psi))  :-
    sat(X,Phi),             % Connect X, Phi
    sat(Y,Psi),             % Connect Y, Psi
    truth_var(X),           % X is in [0,1]
    truth_var(Y),           % Y is in [0,1]
    truth_var(I),           % I is in [0,1]
    control_var(Z),         % Z is in [0,1]
    X + Y + Z $>= I,        % (i)
    X + Y - Z $<= I,        % (ii)
    X + Y - Z $>= 0,        % (iii) see (16)
    X + Y - Z $<= 1,        % (iv)
            I $>= Z.        % (v)

sat(I,neg(Phi))         :- sat(1-I,Phi).

sat(I,atom(P))          :- I $= P.

control_var(0).
control_var(1).

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

Fig. 5. A satisfiability checker for infinite-valued Łukasiewicz logic implemented in Eclipse Prolog

One possibility, is to employ *disjunctive programming* [3, 25] techniques as developed in Operations Research (OR). These give means to combine arbitrary polyhedra disjunctively from their MIP-representations.

Theorem 7 ([25]) *Let $M^1, \ldots, M^t$ be polyhedra in $[0, 1]^k$ with MIP-representations $\mathbf{A}^i\vec{x}^i + \mathbf{B}^i\vec{y}^i \geqslant \mathbf{h}^i$, where in each MIP $\vec{x}^i$ are $[0, 1]$-variables and $\vec{y}^i$ are $\{0, 1\}$-variables. Then the following MIP (where the m_i are new $\{0, 1\}$-variables) is feasible iff $\vec{x} \in (M^1 \cup \cdots \cup M^t)$:*

$$\mathbf{A}^i\vec{x}^i + \mathbf{B}^i\vec{y}^i - \mathbf{h}^i m^i \geq 0 \quad (i = 1, \ldots, t)$$

$$m^1 + \cdots + m^t = 1$$

$$-x_j + x_j^1 + \cdots + x_j^t = 0 \quad (j = 1, \ldots, k)$$

This theorem can be used to compute an MIP-representation as the one of $\oplus$ in (16): MIP-representations of the convex regions whose union constitute the graph of $\oplus$ are obvious from the right part of Fig. 3 — use $x_3^1 = x_1^1 + x_2^1$ and $x_3^2 = 1$. Then apply the theorem to these.

The freshly introduced variables such as the m^i (which play the same rôle as z in (16) are called *control variables*, because they control which disjunction is being selected. Let us close this subsection with a

Proof sketch of theorem 5 part 2 It suffices to construct a McNaughton function which vanishes iff **J** is feasible. To this end, first decompose **J** into a finite union of feasible LPs [25, p. 12]. Each of them describes an m-dimensional convex polyhedron P. Embed this polyhedron into $[0, 1]^{m+1}$ via $(x_1, \ldots, x_m) \mapsto (x_1, \ldots, x_m, 0)$. Now construct piecewise

linear, continuous functions with integral coefficients $f_P : [0, 1]^m \to [0, 1]$ by connecting each corner of P to its nearest point in $\{0, 1\}^m \times \{1\}$. This is possible, because the P are convex. The f_P can obviously be represented by linear polynomials with integral coefficients effectively computed from (the corners of) P. By construction, the f_P are continuous and hence a McNaughton function.

Finally, for each f_P let ϕ_{f_P} be a Łukasiewicz formula as supplied by McNaughton's Theorem. Kleene conjunction $\wedge$ (6) is definable in Łukasiewicz logic and computes the minimum of two truth values, thus the formula $\bigwedge_{f_P} \phi_{f_P}$ takes on the truth value 0 iff the original MIP is feasible. □

McNaughton's Theorem and its variant Theorem 5 express that MIP is captured naturally by Łukasiewicz logic. Earlier I mentioned that classical logic corresponds to IPs of the form (15). Thus the difference between Łukasiewicz logic and classical logic is analogous to that between MIPs and IPs of a very special kind. There is no easy transition between the latter which explains why classical logic is not suitable as a meta language for expressing Łukasiewicz logic.

Another aspect of Theorem 5 part 2 is the following: if every MIP can be expressed as an infinite-valued Łukasiewicz logic formula which in turn can be expressed through an MIP, then the use of the reduction from many-valued logic to MIP developed here seems to be confined to (sublogics of) Łukasiewicz logic. This is not quite true, however: first, it can be used for *any* finite-valued logic $\mathscr{L}$ without the detour of expressing $\mathscr{L}$ in infinite-valued Łukasiewicz logic; second, even if a connective is definable in Łukasiewicz logic, its definition might be very long, whereas its MIP-representation might be short (note that the Corollary does not follow from Theorem 5 part 2 — part 1 of the theorem is required for that); and, most importantly, in the following section I demonstrate that the logics captured by the MIP approach can be extended beyond Łukasiewicz logic.

3.5 Limits and extensions of MIP-representability

The contraposition of Theorem 5.2 yields that the graph of functions not definable in Łukasiewicz logic cannot be MIP-representable. Let us look on the reasons why MIP-representability may fail for a graph of a function f (see [25] for a deepened discussion).

First, f might be not linear or have irrational coefficients. Or infinitely many linear polynomials are required to represent it. Finally, the graph of f can be an open set. For example, it might be not continuous, but composed of finitely many piecewise linear polynomials with integral coefficients. In this case, however, it might be possible to represent the graph of f by MIPs with *strict inequalities*.

A typical example of this situation are Gödel's connectives (see (7, 8) and Fig. 6) which are not continuous.[7] To represent them *strict* linear inequalities or linear *disequations* are needed. An algorithmic treatment of strict inequalities is sometimes possible with so-called transposition theorems [40, p. 94f] which relate strict to non-strict inequalities. More efficient are modified constraint solvers that can directly handle disequations [21, 24].

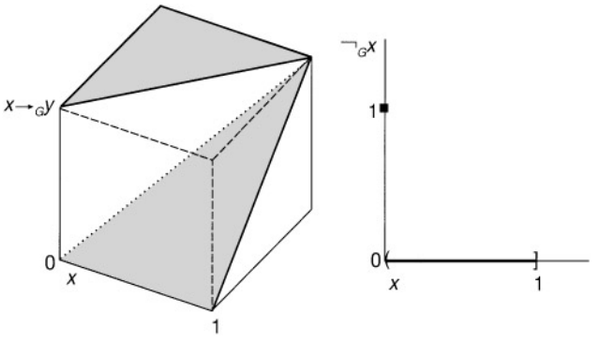

Fig. 6. Function graphs of $x \to_G y$ and $\neg_G x$ on $N = [0, 1]$

It is easy to prove that the strict MIP feasibility is still in NP. If we define strictly MIP-representable logics as MIP-representable logics, where also strict inequations are allowed in the MIP representation, then we have immediately:

Theorem 8 *The* $[c, d]$*-satisfiability problem of any strictly MIP-representable logic for* $0 \leq c \leq d \leq 1$ *is in NP.*

It is known that the satisfiability problems of Łukasiewicz [28] and of Gödel logic [M. Baaz, 1996; Personal Communication] are in NP, but the new technique works even if connectives from both logics are present and it works for altogether different logics as well.

For other interesting, but totally different uses of (M)IP in connection with logic, see [23, 5].

4 Many-valued proof theory, MDDs, logic design: a concordance

For the rest of the paper we restrict attention to finite-valued logics.

4.1 Many-valued analytic cuts

Let us start this part of the paper with a discussion of the cut rule in many-valued logic.

Recall that in classical logic the presence/absence of the cut rule in sequent calculus is equivalent to the presence/absence of the conjunction of all tautologies of the form $\phi \vee \neg \phi$, where ϕ is any formula (if ϕ is restricted to subformulas of the formulas in the goal sequent then one speaks of *analytic cut*).

Hence, formally the cut rule is a DNF representation of truth; as such it can be conceived as a tableau rule schema with empty (i.e. always true) premise:

$$\frac{}{\phi \mid \neg \phi} \tag{17}$$

This rule can be immediately generalized to sets-as-signs DNF representations:

$$\frac{}{S_1 \phi \mid \cdots \mid S_m \phi} \quad m \geq 2, \{S_1, \ldots, S_m\} \text{ set partition of } N \tag{18}$$

[7] It is well-known that Gödel's connectives cannot be expressed in Łukasiewicz logic, so this was to be expected

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

a

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

b

Fig. 7. Different coverings of $\{0, \frac{1}{2}\}$ in truth table of Łukasiewicz sum
a Non-partitioning covering of $\{0, \frac{1}{2}\}$ in truth table of Łukasiewicz sum
b A partitioning covering of $\{0, \frac{1}{2}\}$ in truth table of Łukasiewicz sum

In the truth table in Fig. 7(a) it is shown in an example (corresponding to the rule on the left in (10)) how the union of the extensions of a sets-as-signs rule correspond to a complete covering of the truth table entries that occur in the sign of the premise. This covering is not necessarily a partition, that is, some entries are possibly covered in more than one extension as, for example, it happens in the field containing 0.

With suitable cuts one can enforce that the extensions of a rule form a partition of the entries to be covered.

In Fig. 7(b) a partitioning covering of $\{0, \frac{1}{2}\}$ in the truth table of Łukasiewicz sum is displayed. The rule corresponding to it is as follows:

$$\frac{\{0, \frac{1}{2}\}\ \phi \oplus \psi}{\begin{array}{l|l} \{0, \frac{1}{2}\}\ \phi & \{0\}\ \phi \\ \{0\}\ \psi & \{\frac{1}{2}\}\ \psi \end{array}} \tag{19}$$

Definition 17 Let $\Phi = \bigvee_{r=1}^{M} C_r$ be a sets-as-signs DNF representation of $\phi = S\ \theta(\phi_1, \ldots, \phi_m)$ $(m \geq 1)$. We call Φ a partitioning DNF representation of ϕ iff for any two conjuncts C_i, C_j with $i \neq j$ the set of literals $C_i \cup C_j$ is inconsistent (in the sense of Definition 13). Partitioning sets-as-signs rules are sets-as-signs rules based on a partitioning DNF representation. □

Just as in classical logic with the analytic cut rule (17) it is possible to *derive* many-valued partitioning rules from arbitrary ones with the help of many-valued analytic cut (18).

For instance, with the many-valued cut rule $\overline{\{0\}\ \phi_2 \mid \{\frac{1}{2}, 1\}\ \phi_2}$ one derives (19) from the rule on the left in (10). One first applies the cut rule and then in each extension (10). Finally one gets rid of inconsistent and subsumed branches.

4.2
Many-valued decision diagrams

Binary decision diagrams (BDDs) and their relatives are a family of data structures originally developed for efficient representation and manipulation of Boolean formulas, but more recently also used to represent polynomials, finite domain functions, or finite sets. The standard reference for BDDs is [10], a survey of the field is contained in [11] and, more recently and exhaustively, in [38]. An introduction to BDDs intended for the automated theorem proving community is [45].

One strength of BDDs is that they can represent the models of very large satisfiable formulas in an efficient manner. Moreover, insertion of new formulas and combination of BDDs can be done quickly as well. There exist very efficient packages for BDD manipulation [7] whose use is quite popular in logic design, see e.g. [12].

Basically, BDDs are a representation of Boolean functions based on the three-place if-then-else connective:

$$\text{if } i \text{ then } j \text{ else } k = (i \wedge j) \vee (\neg i \wedge k) = \begin{cases} j & \text{if } i = 1 \\ k & \text{if } i = 0 \end{cases}$$

Every Boolean function can be expressed with a formula that contains no connective but if-then-else, logical constants 0 and 1, and where atomic formulas occur exactly as the first arguments of the if-then-else connectives. For instance, $p \wedge q$ is equivalent to

$$\text{if } p \text{ then (if } q \text{ then 1 else 0) else 0.}$$

Such a representation of a formula is called an if-then-else normal form, BDD, or Shannon tree. A systematic way to obtain a BDD representation of a formula or logical function ϕ is provided by the so-called *Shannon expansion*.[8] Assume that the atoms occurring in ϕ are $\{p_1, \ldots, p_m\}$ and denote this with $\phi(p_1, \ldots, p_m)$. Then

$$\begin{aligned} \phi(p_1, p_2, \ldots, p_m) = &\text{if } p_1 \text{ then } \phi(1, p_2, \ldots, p_m) \\ &\text{else } \phi(0, p_2, \ldots, p_m) \\ = &(\{1\}\, p_1 \wedge \phi(1, p_2, \ldots, p_m)) \vee \\ &(\{0\}\, p_1 \wedge \phi(0, p_2, \ldots, p_m)) \end{aligned} \tag{20}$$

Recursive application of (20) and replacing variable-free formulas with their function value obviously gives a BDD representation.

Usually, BDDs are assumed to be reduced and ordered (then abbreviated ROBDD). Reduced means that the syntactic tree of a BDD is turned into a graph by identifying isomorphic subtrees and applying the following simplication rule wherever possible:

$$(\text{if } i \text{ then } j \text{ else } j) = j.$$

Ordered means that relative to a given total ordering $\prec$ on atoms, whenever q occurs in the body of "if p ..." then $p \prec q$ must hold. An important property of ROBDDs is that two ROBDDs of the same Boolean function are identical up to isomorphism rendering them a *strong normal form* for Boolean functions.

The relevance of BDDs for our discussion comes from the facts that first, there is a close relationship between BDDs and tableaux with partitioning rules [33] and second, they can be extended to *many-valued decision diagrams* and finite-valued logics in a natural way by simply replacing the if-then-else with an $(n+1)$-ary case-of connective in n-valued logic:

$$\begin{array}{l} \text{case } i \text{ of} \\ \quad 0, : j_0; \\ \quad \frac{1}{n-1} : j_1; \\ \quad \ldots \quad \ldots \\ \quad 1 : j_{n-1} \\ \text{esac} \end{array} = \begin{cases} j_0 & \text{if } i = 0 \\ j_1 & \text{if } i = \frac{1}{n-1} \\ \ldots & \ldots \\ j_{n-1} & \text{if } i = 1 \end{cases}$$

[8] In the BDD and function minimization literature this equation is usually attributed to Shannon [41] or Akers [1], however, it appears already in [6]. Expansions are sometimes called *decompositions*

Orłowska [30] gave a proof procedure for propositional Post logic based on an MDD-like structure, but she used a different (and slightly cryptic) notation. MDDs were rediscovered in connection with the growing interest in BDD methods in [44], where it is also shown that like their binary counterparts n-valued MDDs are functionally complete, they can be computed with the help of a generalized Shannon expansion

$$\phi(p_1, \dots, p_m) = \begin{cases} \text{case } i \text{ of} \\ \quad 0 : \phi(0, p_2, \dots, p_m); \\ \dfrac{1}{n-1} : \phi\left(\dfrac{1}{n-1}, p_2, \dots, p_m\right); \\ \quad \dots \qquad \dots \\ \quad 1 : \phi(1, p_2, \dots, p_m) \\ \text{esac} \end{cases} \tag{21}$$

$$= \bigvee_{i \in N} (\{i\}\, p_1 \wedge \phi(i, p_2, \dots, p_m))$$

and that ROMDDs (called *canonical function graphs* in [44]) are a strong normal form representation of arbitrary n-valued functions.

As already mentioned, BDDs and tableaux with partitioning rules (Definition 17) bear a close relationship. Consider the partitioning tableau rule and BDD for classical disjunction (which can also be seen as Kleene disjunction when $n=2$) depicted in Figs. 8(a) and (b).

In the BDD edges corresponding to then and else branches are labeled with 1 and 0, respectively. An edge labeled with i that comes out of a node p can be seen as an assertion of the truth value i to p, in other words a signed formula $\{i\}\, p$. Now the following relationship between tableaux with partitioning rules and (RO)BDDs holds (cf. [33]): for each set of signed literals corresponding to the edges on a path in a (RO)BDD for ϕ that ends with 1 there is an open branch in any tableau with partitioning rules for $\{1\}\ \phi$ containing exactly the same literals and vice versa.

This relation extends to singleton signs tableaux with partitioning rules and (RO)MDDs in the following way: for each set of signed literals corresponding to the edges on a path in a (RO)MDD for ϕ that ends with j there is an open branch in any singleton sets tableau with partitioning rules for $\{j\}\ \phi$ containing exactly the same literals and vice versa. For instance, an RMDD for $\{\frac{1}{2}\}\ (p \oplus q)$ (three-valued Łukasiewicz sum (2)), is displayed in Fig. 8(d); the paths ending with $\frac{1}{2}$ correspond to the extensions of the partitioning tableau rule in Fig. 8(c).

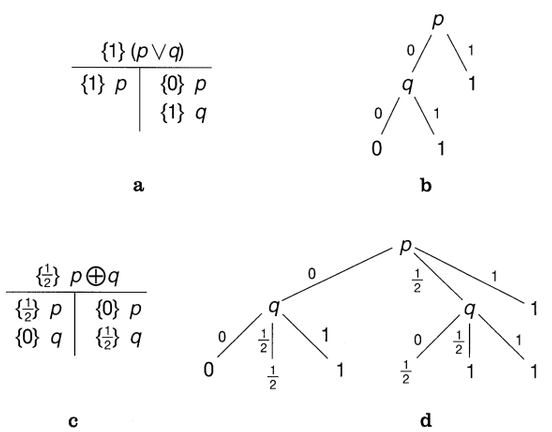

Fig. 8a–d. Signed tableaux with partitioning rules vs. BDDs and MDDs

A BDD or MDD representation (or the literals on the open branches of a partitioning tableau) can be seen as a Boolean polynomial over signed literals with truth values as coefficients. Let us write a signed literal of the form $\{i\}\ p$ as p^i, " $\wedge$ " as ".", " $\vee$ " as "$+$". Then, by (20), for example,

$$\begin{aligned} p \vee q &= p^1 \cdot (1 \vee q) + p^0 \cdot (0 \vee q) \\ &= p^1 \cdot 1 + p^0 \cdot (q^1 \cdot (0 \vee 1) + q^0 \cdot (0 \vee 0)) \\ &= p^1 \cdot 1 + p^0 \cdot q^1 \cdot 1 + p^0 \cdot q^0 \cdot 0 \end{aligned}$$

Polynomial representations can be generalized. In the many-valued case one may, of course, consider arbitrary signed literals of the form $S\ p$. Written as p^S they are well-known in logic design [35], sometimes under the name set literal or universal literal.[9] On the other hand, there is no reason to restrict oneself to unary functions for the base of a polynomial representation. Of course one needs to make restrictions lest the resulting representations are useless. Equation (21), for example, might be generalized to

$$\phi(p_1, \dots, p_j, \dots, p_m) = \sum_{i \in N} \Psi_i \cdot \phi(p_1, \dots, i, \dots, p_m) \tag{22}$$

for a certain base of Boolean functions $\{\Psi_0, \dots, \Psi_1\}$. Examples of generalized expansions in the Boolean case are the *orthonormal expansions* of Löwenheim [26], see also [9]. Expansions for many-valued logic have been suggested and investigated in several papers, for example, [36, 4]. Orthonormal expansions were recently generalized to many-valued logic by Perkowski [31] in an attempt to systematize the plethora of existing expansions.

In logic design one often needs to synthesize circuits based on the *exclusive* or *parity* connective $\oplus$ (not to be confused with Łukasiewicz sum) rather than disjunction $\vee$ [38, Chapter 2]. Here, one may use expansions based on $\oplus$ called *positive* and *negative Davio expansion*:

$$\begin{aligned} &\phi(p_1, p_2, \dots, p_m) \\ &= \phi(0, p_2, \dots, p_m) \oplus p_1^1 \cdot (\phi(0, p_2, \dots, p_m) \\ &\quad \oplus \phi(1, p_2, \dots, p_m)) \\ &= \phi(1, p_2, \dots, p_m) \oplus p_1^0 \cdot (\phi(0, p_2\ \dots, p_m) \\ &\quad \oplus \phi(1, p_2, \dots, p_m)) \end{aligned} \tag{23}$$

Davio expansions, like Shannon expansions, when denoted as trees lead to a another class of decision diagrams

[9] In logic design methods often restrictions on the form of S are imposed, for example, a popular class of literals are *window literals*, where S is of the form $[i, j]$ with $i \leq j$. Recently, however, the use of completely general literals has been advocated [14]

(so-called Kronecker decision diagrams, see e.g. [38, Chapter 2]). They can be generalized to many-valued logic and even to set literals [31, 36]:

$$\phi(p_1, p_2, \ldots, p_m) = \bigoplus_{i \in N} (p_1^{S_i} \cdot \psi(i, p_2, \ldots, p_m)) \tag{24}$$

for suitable sets $S_i \subseteq N$ and a function ψ depending on the S_i.

Certain orthonormal expansions based on unary functions result in partitioning sets-as-signs rules (and hence many-valued cuts) as introduced in Definition 17. More general expansions have no representation in rule form; on the other hand, arbitrary sets-as-signs tableau rules do not necessarily correspond to any expansion of the form (22). The reason for this is that tableau rules work by analyzing the leading connective of a complex formula, whereas expansion schemes as used in MDDs and logic synthesis work by analyzing a certain variable of the formula. Through many-valued cut rules both schemata are linked. The exact relationship, however, remains to be investigated.

4.3 Logic synthesis

In this section the link between proof theory and logic design is strengthened by demonstrating that OR-AND-OR implementations can be synthesized via sets-as-signs tableaux. As background reading on logical circuit synthesis [8] is suggested.

Example 4 ([37, Chapter 12]) Let a two-valued four-place function $f(z, w, x, y)$ be defined as in the Karnaugh map in Fig. 9(a).

It is possible to interpret f as a *four-valued two-place function* $f(Y, X)$ (displayed in Fig. 9(c)) via the variable mapping given in Fig. 9(b). A minimal sets-as-signs DNF tableau rule for $\{1\}\ f(Y, X)$ is easily obtained.

$\{1\}\ f(Y,X)$			
$\{0, \frac{1}{3}, \frac{2}{3}\}\ X$	$\{\frac{2}{3}, 1\}\ X$	$\{\frac{1}{3}\}\ X$	$\{0, 1\}\ X$
$\{0\}\ Y$	$\{\frac{1}{3}\}\ Y$	$\{\frac{2}{3}\}\ Y$	$\{1\}\ Y$

Decoding the variables in the extensions into (z, w, x, y) one obtains:

$1\ f(z,w,x,y)$			
$0\,x \vee 1\,y$	$1\,x$	$0\,x$	
		$1\,y$	$0\,y$
$0\,z$	$0\,z$	$1\,z$	$1\,z$
$0\,w$	$1\,w$	$1\,w$	$0\,w$

This rule, however, can directly serve as a specification for the OR-AND-OR circuit displayed in Fig. 10. □

In general, the specification of a function f can be nested, thus several rule applications may be needed. The structure of the open branches of the resulting tableau gives the AND-OR or PLA (see below) part of the circuit, while the signed

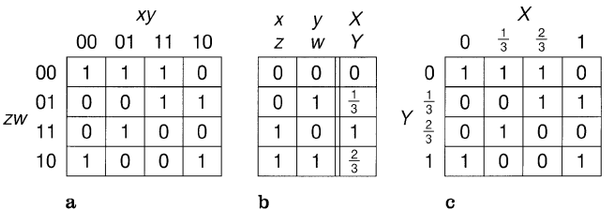

Fig. 9a–c. Function from previous example as Karnaugn map and four-valued truth table. **a** Karnaugh map of f, **b** variable mapping, **c** four-valued truth table

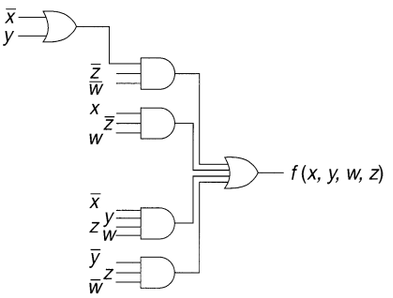

Fig. 10. OR-AND-OR circuit

literals in the extensions represent another level of ORs (only one OR gate is needed in the example). Instead of a whole level of ORs one can also supply a fixed structure: an n-bit decoder (in the example n is 2) mapping a group of n two-valued variables to an 2^n-valued variable. This class of circuits is also known as *programmable logic arrays* (PLA) with n-bit decoder.

Sasao [35] showed that AND-OR realizations with decoders and, to an even greater extent, OR-AND-OR realizations can be considerably smaller than mere AND-OR realizations. As is to be expected, among other factors the choice of the mapping between binary and many-valued variables influences the quality of the result.

Although some parallels and correspondences between many-valued calculi and methods from logic design were exhibited, in both areas rather different goals are being pursued: Methods developed for obtaining minimal representations of many-valued functions in logic design often have a heuristic[10] nature which means they yield in general only a near minimal solution; on the other hand, they can deal with rather large inputs. The functions to be modeled as a circuit are, of course, typically not constant, hence neither tautologies nor unsatisfiable. And finally, the specification of logic design problems is almost always purely propositional. Still, I think it would be fruitful if both fields became better aware of each other. To roundoff this section I summarise different terms and notation used by the different communities in Table 1.

[10] There are exact methods, though: see, for example, [37, Chapter 1]

Table 1. Different terms and notation for concepts in proof theory and logic design

Proof Theory & Automated Theorem Proving	Logic Design & Function Minimization
Truth value set $N=\left\{0, \frac{1}{n-1}, \ldots, \frac{n-2}{n-1}, 1\right\}$	$P=\{0, 1, \ldots, p-2, p-1\}$, particularly $B=\{0, 1\}$
Connective $A(\theta): N^k \to N$	Function $f: P_1 \times \cdots \times P_k \to P_i$ $P_i \neq P_j$ possible (i.e. mixed-radix)
Boolean connective $A(\theta): \{0, 1\}^k \to \{0, 1\}$	Switching function $f: B^k \to B$
Cardinality n of N	Radix r of function f, i.e. $r=\max_{1 \leq i \leq k}\lvert P_i\rvert$
Interpretation (element of N^k)	Minterm $(x_1, \ldots, x_k) \in P_1 \times \cdots \times P_k$
Literal p, $\neg p$ Signed literal $l=Sp$	x, $\bar{x}$ Set literal $X=x^S$
Conjunct, conjunctive clause $C=l_1 \wedge \cdots \wedge l_m$	Product term $P=X_1 X_2 \cdots X_m$
Disjunctive normal form, DNF $\bigvee_i C_i = C_1 \vee \cdots \vee C_m$; models of $\bigvee_i C_i$	Sum-of-products, SOP $\sum_i P_i = P_1 + \cdots + P_m$; minterms contained in $\sum_i P_i$
Tableau rule/DNF representation for θ Sets-as-signs/DNF representation for θ	SOP expression of $f=A(\theta)$ SOP expression over set literals
Antivalence, exclusive or, $\not\leftrightarrow$, $\not\equiv$	EXOR, XOR, $\oplus$
Proof by case distinction, cut if-then-else connective	Expansion, decomposition Shannon expansion
Certain partitioning rules	Certain general expansions
Literals on completed, open tableau branch/on non-axiomatic end sequent	1-branch of BDD
if-then-else normal form case-of normal form	Binary decision diagram, BDD Many-valued decision diagram, MDD

References

1. **Akers, S.B.:** Binary decision diagrams. IEEE Transactions on Computers, 27(6): 509–516, June 1978
2. **Baaz, M.; Fermüller, C.G.:** Resolution-based theorem proving for many-valued logics. Journal of Symbolic Computation, 19(4): 353–391, Apr. 1995
3. **Balas, E.:** Disjunctive programming. Annals of Discrete Mathematics, 5 (Discrete Optimization II): 3–51, 1979. Proc. of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium, Canada 1977. Edited by P.L. Hammer E.L. Johnson, and B.H. Korte
4. **Becker, B.; Drechsler, R.:** Efficient graph-based representation of multi-valued functions with an application to genetic algorithms. In Proc. 24th International Symposium on Multiple-Valued Logic, Boston/MA, pages 65–72. IEEE Press, Los Alamitos, May 1994
5. **Bell, C.; Nerode, A.; Ng, R.; Subrahmanian, V.:** Mixed integer programming methods for computing nonmonotonic deductive databases. Journal of the ACM, 41(6): 1178–1215, Nov. 1994
6. **Boole, G.:** An Investigation of the Laws of Thought. Walton, London, 1854. Reprinted by Dover Books, New York, 1954
7. **Brace, K.S.; Rudell, R.L.; Bryant, R.E.:** Efficient implementation of a BDD package. In Proc. 27th ACM/IEEE Design Automation Conference, pages, 40–45. IEEE Press, Los Alamitos, 1990
8. **Brayton, R.K.; Hachtel, G.D.; McMullen, C.T.; Sangiovanni-Vincentelli, A.L.:** Logic Minimization Algorithms for VLSI Synthesis. Kluwer, Boston, 1984
9. **Brown, F.M.:** Boolean Reasoning. Kluwer, Norwell/MA, USA, 1990
10. **Bryant, R.E.:** Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35: 677–691, 1986
11. **Bryant, R.E.:** Symbolic boolean manipulation with ordered binary decision diagrams. ACM Computin Surveys, 24(3): 293–318, Sept. 1992
12. **Burch, J.; Clarke, E.; McMillan, K.; Dill, D.:** Sequential circuit verification using symbolic model checking. In Proc. 27th Design Automation Conference (DAC 90), pages 46–51, 1990
13. **Carnielli, W.A.:** Systematization of finite many-valued logics through the method of tableaux. Journal of Symbolic Logic, 52(2): 473–493, June 1987
14. **Dueck, G.W.; Butler, J.T.:** Multiple-valued logic operations with universal literals. In Proc. 24th International Symposium on Multiple-Valued Logic, Boston/MA, pages 73–79. IEEE Press, Los Alamitos, May 1994
15. **Fitting, M.C.:** First-Order Logic and Automated Theorem Proving. Springer-Verlag, New York, second edition, 1996
16. **Hähnle, R.:** Towards an efficient tableau proof procedure for multiple-valued logics. In E. Börger, H. Kleine Büning, M.M. Richter, and W. Schönfeld, editors, Selected Papers from Computer Science Logic, CSL'90, Heidelberg, Germany, volume 533 of LNCS, pages 248–260. Springer-Verlag, 1991
17. **Hähnle, R.:** Automated Deduction in Multiple-Valued Logics, volume 10 of International Series of Monographs on Computer Science. Oxford University Press, 1994
18. **Hähnle, R.:** Many-valued logic and mixed integer programming. Annals of Mathematics and Artificial Intelligence, 12(3, 4): 231–264, Dec. 1994
19. **Hähnle, R.:** Short conjunctive normal forms in finitely -valued logics. Journal of Logic and Computation, 4(6): 905–927, 1994
20. **Hähnle, R.:** Exploiting data dependencies in many-valued logics. Journal of Applied Non-Classical Logics, 6(1): 49–69, 1996

21. **Hentenryck, P.V.; Graf, T.:** Standard Forms for Rational Linear Arithmetics in Constraint Logic Programming. Annals of Mathematics and Artificial Intelligence, 5(2–4), 1992
22. **Hooker, J.N.:** A quantitative approach to logical inference. Decision Support Systems, 4: 45–69, 1988
23. **Hooker, J.N.:** New methods for computing inferences in first order logic. Annals of Operations Research, 43(1–4): 479–492, Oct. 1993. Selected Papers of Applied Mathematical Programming and Modelling, APMOD91
24. **Imbert, J.L.; Hentenryck, P.V.:** Efficient Handling of Disequations in CLP over Linear Rational Arithmetics. In F. Benhamou and A. Colmerauer, editors, Constraint Logic Programming: Selected Research, pages 49–71. MIT Press, 1993
25. **Jeroslow, R.G.:** Logic-Based Decision Support. Mixed Integer Model Formulation. Elsevier, Amsterdam, 1988
26. **Löwenheim, L.:** Über die Auflösung von Gleichungen im logischen Gebietekalkül. Mathematische Annalen, 68: 169–207, 1910
27. **McNaughton, R.:** A theorem about infinite-valued sentential logic. Journal of Symbolic Logic, 16(1): 1–13, 1951
28. **Mundici, D.:** Satisfiability in many-valued sentential logic is NP-complete. Theoretical Computer Science, 52: 145–153, 1987
29. **Mundici, D.:** A constructive proof of McNaughton's Theorem in infinite-valued logic. Journal of Symbolic Logic, 59(2): 596–602, June 1994
30. **Orlowska, E.:** Mechanical proof procedure for the *n*-valued propositional calculus. Bull. de L'Acad. Pol. des Sci., Série des sci. math., astr. et phys., XV(8): 537–541, 1967
31. **Perkowski, M.A.:** The generalized orthonormal expansion of functions with multiple-value inputs and some of its application. In Proc. 22nd International Symposium on Multiple-Valued Logic, pages 442–450. IEEE Press, Los Alamitos, May 1992
32. **Plaisted, D.A.; Greenbaum, S.:** A structure-preserving clause form translation. Journal of Symbolic Computation, 2: 293–304, 1986
33. **Posegga, J.:** Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe. PhD thesis. University of Karlsruhe, 1993. diski 51, infix Verlag
34. **Rousseau, G.:** Sequents in many valued logic I. Fundamenta Mathematicae, LX: 23–33, 1967
35. **Sasao, T.:** Multiple-valued decomposition of generalized Boolean functions and the complexity of programmable logic arrays. IEEE Transactions on Computers, C-30: 635–643, Sept. 1981
36. **Sasao, T.:** Optimization of multi-valued AND-XOR expressions using multiple-place decision diagrams. In Proc. 22nd International Symposium on Multiple-Valued Logic, pages 451–458, IEEE Press, Los Alamitos, May 1992
37. **Sasao, T.:** Logic Synthesis and Optimization. Kluwer, Norwell/MA, USA, 1993
38. **Sasao, T.; Fujita, M.:** (eds). Representations of Discrete Functions. Kluwer Academic Publishers, Boston, 1996
39. **Scarpellini, B.:** Die Nichtaxiomatisierbarkeit des unendlichwertigen Prädikatenkalküls von Łukasiewicz. Journal of Symbolic Logic, 27(2): 159–170, June 1962
40. **Schrijver, A.:** Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, 1986
41. **Shannon, C.E.:** A symbolic analysis of relay and switching circuits. AIEE Transactions, 67: 713–723, 1938
42. **Siekmann, J.; Wrightson, G.:** (eds). Automation of Reasoning: Classical Papers in Computational Logic 1967-1970, volume 2. Springer-Verlag, 1983
43. **Smullyan, R.M.:** First-Order Logic. Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer-Verlag
44. **Srinivasan, A.; Kam, T.; Malik, S.; Brayton, R.E.:** Algorithms for discrete function manipulation. In Proc. IEEE International Conference on CAD, Santa Clara/CA, USA, pages 92–95. IEEE Press, Los Alamitos, Nov. 1990
45. **Strother Moore, J.:** Introduction to the OBDD algorithm for the ATP community. Journal of Automated Reasoning, 12(1): 33–45, 1994
46. **Takahashi, M.:** Many-valued logics of extended Gentzen style I. Science Reports of the Tokyo Kyoiku Daigaku, Section A, 9(231): 95–116, 1967
47. **Tseitin, G.:** On the complexity of proofs in propositional logics. Seminars in Mathematics, 8, 1970. Reprinted in [42]
48. **Zach, R.:** Proof theory of finite-valued logics. Master's thesis, Institut für Algebra und Diskrete Mathematik, TU Wien, Sept. 1993. Available as Technical Report TUW-E185.2-Z.1-93