Łukasiewicz Logic: From Proof Systems To Logic Programming

GEORGE METCALFE[1], *Institute of Discrete Mathematics and Geometry, Technical University Vienna, Wiedner Hauptstrasse 8-10, A-1040 Vienna, Austria. Email: metcalfe@logic.at*

NICOLA OLIVETTI, *Department of Computer Science, University of Turin, Corso Svizzera 185, 10149 Turin, Italy. Email: olivetti@di.unito.it*

DOV GABBAY, *Department of Computer Science, King's College London, Strand, London WC2R 2LS, UK. Email: dg@dcs.kcl.ac.uk*

Abstract

We present logic programming style "goal-directed" proof methods for Łukasiewicz logic **Ł** that both have a logical interpretation, and provide a suitable basis for implementation. We introduce a basic version, similar to goal-directed calculi for other logics, and make refinements to improve efficiency and obtain termination. We then provide an algorithm for fuzzy logic programming in Rational Pavelka logic **RPL**, an extension of **Ł** with rational constants.

Keywords: Łukasiewicz Logics, Fuzzy Logics, Goal-Directed Methods, Logic Programming

1 Introduction

Łukasiewicz logics were introduced by Jan Łukasiewicz in the 1920s [15], and are important in several areas of research. In fuzzy logic, the infinite-valued Łukasiewicz logic **Ł**, along with Gödel logic **G** and Product logic **Π**, emerges as one of the fundamental "t-norm based" fuzzy logics used for reasoning about vagueness [9]. In algebra, Chang's MV-algebras for **Ł** have applications in many fields of mathematics; for a comprehensive study see [2]. In geometry, formulae of **Ł** are related to particular geometric functions via McNaughton's representation theorem [16]. Finally, various semantic interpretations of Łukasiewicz logics have been given; e.g. via Ulam's game with errors/lies which has applications to adaptive error-correcting codes [23].

A variety of proof methods have been defined for theorem proving in **Ł**. These fall into three main categories:

1. *Gentzen-style calculi.* Sequent and hypersequent calculi for **Ł** have been introduced by Metcalfe et al. [19]. Also, a many-placed sequent calculus for **Ł** (via a reduction to finite-valued Łukasiewicz logics) has been given by Aguzzoli and Ciabattoni [1].
2. *Tableaux systems.* Hähnle [8] and Olivetti [25] have defined tableaux for **Ł** via reductions to mixed integer programming problems. Both provide co-NP decision

[1]Research supported by Marie Curie Fellowship 501043.

procedures for checking validity in **Ł**.

3. *Resolution methods.* Wagner [32] (using calculation of intersecting hyperplanes), and Mundici and Olivetti [24] (using calculation of Θ-supports of formulae) have defined resolution methods for **Ł**.

Connectives from **Ł** have also been used as the basis for *fuzzy logic programming* methods, see e.g. [13, 31, 30]. In fuzzy logic programming, and other "quantitative" variants/extensions of PROLOG such as Annotated Logic Programs [12], Residuated Logic Programs [3] and Possibilistic Logic Programs [4], the standard machinery of classical logic programming is adapted to handle uncertain or vague information. (Horn) clauses and facts have numerical data attached expressing a confidence factor or a degree of truth; for example, clauses can be of the form

$$TV : Body(\bar{X}) \to Head(\bar{X})$$

where TV denotes a truth value, and $Head$ and $Body$ are, respectively, an atomic formula, and a combination of atomic formulae in the (universally quantified) variables $\bar{X}$. The combining connectives in $Body$ and also the implication $\to$ are often interpreted by t-norms and their residua, e.g. in the case of Łukasiewicz logic programming, the Łukasiewicz t-norm and its residuum. The aim is then to compute the highest truth-value with which a goal is entailed, e.g. using classical (first order) resolution extended with some constraint propagation mechanism.[2] Note that for examples of Łukasiewicz logic programming in the literature, performing this computation is very different to theorem proving in **Ł**. Instead of checking the *validity* of a given formula, the computation checks to what degree an atomic formula is the *logical consequence* of formulae in a restricted language (i.e. Horn clauses).

In this paper[3] our aim is to define new proof systems for **Ł** and an extension of **Ł** with rational constants called *Rational Pavelka logic* **RPL** [26] (see also [9]) that are both suitable for automated reasoning, and have intuitive logical and algorithmic interpretations. These proof systems should then provide a basis for both theorem proving and fuzzy logic programming in Łukasiewicz logics. That is to say, calculi for theorem-proving on the one hand, and for logic programming on the other, are obtained as refinements of a purely logical calculus. Our strategy for developing such systems is to make use of theoretical insights provided by the Gentzen-style calculi of [19] to develop *goal-directed methods* for **Ł**. Goal-directed methods (often developed using the Uniform Proof paradigm of Miller [21]) are a generalization of the logic programming style of deduction particularly suitable for proof search. Goal (succedent) formulae are decomposed according to their structure, while database (antecedent) formulae are decomposed according to the goals, thereby eliminating much non-determinism and avoiding decomposing irrelevant formulae. Goal-directed systems have been given for a wide range of logics including classical, intuitionistic, intermediate, modal, substructural and many-valued logics [21, 5, 6, 17], and have been used as the basis for various non-classical logic programming languages, see e.g. [27].

We proceed as follows. In Section 2 we review basic definitions and results for **Ł** and **RPL**, and in Section 3 we give a brief introduction to goal-directed methods. In

[2] More precisely, the derivation of a goal is performed by combining SLD-resolution steps with the backward calculation of truth values using the truth functions associated to connectives.

[3] A shorter version, omitting several proofs and the extension to fuzzy logic programming, appeared in [18].

Section 4 we define a basic goal-directed calculus for **Ł** based on the language consisting only of implication $\rightarrow$ and $\overline{0}$. In Section 5 we turn our attention to improving efficiency, and obtaining termination for the calculus. In Section 6 we adapt our basic calculus to handle further connectives, rational constants, and logical consequence, thereby obtaining the basis for an algorithm for fuzzy logic programming in **RPL**.

2 Łukasiewicz Logic and Rational Pavelka Logic

In this section we give a brief overview of some of the main definitions and results for propositional Łukasiewicz logic **Ł** and its extension Rational Pavelka logic **RPL**.

Definition 2.1 (Łukasiewicz Logic **Ł**)
Łukasiewicz logic **Ł** is based on a language with a binary connective $\rightarrow$, a constant $\overline{0}$, and defined connectives:[4]

$$\begin{array}{rcl@{\qquad\qquad}rcl} \neg A & =_{def} & A \rightarrow \overline{0} & A \odot B & =_{def} & \neg(A \rightarrow \neg B) \\ A \wedge B & =_{def} & A \odot (A \rightarrow B) & A \vee B & =_{def} & (A \rightarrow B) \rightarrow B \end{array}$$

A *valuation* for **Ł** is a function v from the set of propositional variables to $[0,1]$, extended to formulae by the conditions:

$$v(\overline{0}) = 0 \quad \text{and} \quad v(A \rightarrow B) = min(1, 1 - v(A) + v(B))$$

A formula A is *valid* in **Ł**, written $\models_{\mathbf{Ł}} A$, iff $v(A) = 1$ for all valuations v for **Ł**.

A is a *logical consequence* of a set of formulae Γ in **Ł**, written $\Gamma \models_{\mathbf{Ł}} A$, iff for any valuation v for **Ł**, whenever $v(B) = 1$ for all $B \in \Gamma$, also $v(A) = 1$.

Two formulae A and B are *logically equivalent*, written $A \equiv_{\mathbf{Ł}} B$, iff $v(A) = v(B)$ for all valuations v for **Ł**.

By calculation, for all valuations v:

$$\begin{array}{rcl@{\qquad\qquad}rcl} v(\neg A) & = & 1 - v(A) & v(A \odot B) & = & max(0, v(A) + v(B) - 1) \\ v(A \wedge B) & = & min(v(A), v(B)) & v(A \vee B) & = & max(v(A), v(B)) \end{array}$$

The following axiomatization for **Ł** (together with a fifth axiom later proved to be redundant) was originally proposed by Łukasiewicz, and proved complete first by Wajsberg (unpublished proof), and subsequently by Rose and Rosser [28].

$$\begin{array}{ll} (\text{Ł1}) & A \rightarrow (B \rightarrow A) \\ (\text{Ł2}) & (A \rightarrow B) \rightarrow ((B \rightarrow C) \rightarrow (A \rightarrow C)) \\ (\text{Ł3}) & ((A \rightarrow B) \rightarrow B) \rightarrow ((B \rightarrow A) \rightarrow A) \\ (\text{Ł4}) & ((A \rightarrow \overline{0}) \rightarrow (B \rightarrow \overline{0})) \rightarrow (B \rightarrow A) \end{array}$$

$$(mp) \quad \frac{A \rightarrow B, A}{B}$$

The appropriate complexity class for **Ł** was determined by Mundici in [22].

[4]Note that $\wedge$ and $\vee$ are weak conjunction and disjunction respectively, while $\odot$ is a strong conjunction interpreted by the Łukasiewicz t-norm.

Theorem 2.2 (Mundici [22])
The validity problem for **Ł** is co-NP-complete.

Rational Pavelka logic **RPL** is obtained by adding truth constants for all the rational numbers in the $[0,1]$ interval [26].[5]

Definition 2.3 (Rational Pavelka Logic **RPL**)
Rational Pavelka Logic **RPL** is based on a language with a binary connective $\to$, a set of constants $\overline{C} = \{\overline{r} \ : \ r \in [0,1] \cap \mathbb{Q}\}$ and the same defined connectives as **Ł**, where valuations v for **RPL** are as for **Ł**, but with $v(\overline{r}) = r$ for all $\overline{r} \in \overline{C}$. Notions of validity, logical consequence, and logical equivalence, are extended to **RPL** in the obvious way, and use the same notation.

An axiomatization for **RPL** is obtained by extending the above system for **Ł** with bookkeeping axioms for truth constants, i.e.

$$((\overline{r} \to \overline{s}) \to \overline{min(1, 1-r+s)}) \wedge (\overline{min(1, 1-r+s)} \to (\overline{r} \to \overline{s})) \quad \text{for } \overline{r}, \overline{s} \in \overline{C}$$

Unlike classical logic and in common with other substructural logics, **Ł** and **RPL** may be thought of as having *two* different consequence relations, one defined by logical consequence, and the other by implication. In classical logic, the two relations are equivalent. In **Ł** the connection between the two notions is given by the following version of the deduction theorem.

Theorem 2.4 (Rose and Rosser [28])
$\Gamma, A \models_{\mathbf{Ł}} B$ iff $\Gamma \models_{\mathbf{Ł}} (\underbrace{A \odot \ldots \odot A}_{n}) \to B$ for some $n \in \mathbb{N}$.

3 Goal-Directed Methods

In this paper we adopt a "logic programming style" goal-directed paradigm of deduction. For a given logic, denote by $\Gamma \vdash^? A$ the query "does A follow from Γ?" where Γ is a database (collection) of formulae and A is a goal formula. The deduction is *goal-directed* in the sense that the next step in a proof is determined by the form of the current goal. A complex goal is decomposed until its atomic constituents are reached. An atomic goal q is then matched (if possible) with the "head" of a formula $G \to q$ in the database, and its "body" G asked in turn. This can be viewed as a sort of resolution step, or generalized Modus Tollens.

Goal-directed deduction can be refined in several ways. (1) by putting constraints or labels on database formulae, restricting those available to match an atomic goal. (2) by adding more control to ensure termination, e.g. by loop-checking, or "diminishing resources" i.e. removing formulae "used" to match an atomic goal. (3) by re-asking goals previously occurring in the deduction using *restart rules*. However, note that for applications such as deductive databases and logic programming, a terminating proof procedure is not always essential. We might want to get a proof of the goal from the database quickly if one exists, but be willing to have no answer otherwise (and pre-empt termination externally).

Goal-directed procedures have been proposed for a variety of logics. We illustrate the methodology here by presenting a diminishing resources with bounded restart

[5] Note that by using only rational numbers, the language remains countable.

algorithm given in [5] for the implicational fragment of intuitionistic logic.[6] We adopt the convention of writing $\{A_1, \ldots, A_n\} \rightarrow q$ for $A_1 \rightarrow (A_2 \rightarrow \ldots (A_n \rightarrow q) \ldots)$, where $Head(\{A_1, \ldots, A_n\} \rightarrow q) = q$ and $Head(\Gamma) = \{Head(A) \ : \ A \in \Gamma\}$.

Definition 3.1 (**GDLJ**$^{\rightarrow}$)
Queries for **GDLJ**$^{\rightarrow}$ have the form $\Gamma \vdash^? G; H$ where Γ is a set of formulae called the database, G is a formula called the goal, and H is a sequence of atomic goals called the history. The rules are as follows (where $H * (q)$ is q appended to H):

(*success*)	$\Gamma \vdash^? q; H$ succeeds if $q \in \Gamma$
(*implication*)	From $\Gamma \vdash^? \Pi \rightarrow q; H$ step to $\Gamma, \Pi \vdash^? q; H$
(*reduction*)	From $\Gamma, \Pi \rightarrow q \vdash^? q; H$ step to $\Gamma \vdash^? A; H * (q)$ for all $A \in \Pi$
(*bounded restart*)	From $\Gamma \vdash^? q; H$ step to $\Gamma \vdash^? p; H * (q)$ if p follows q in H

Example 3.2
Consider the following proof, observing that (*bounded restart*) is needed at (2) to compensate for the removal of $p \rightarrow q$ at (1):

$$\begin{array}{llcll}
 & & \vdash^? & [(p \rightarrow q) \rightarrow p, p \rightarrow q] \rightarrow q; \emptyset & (\textit{implication}) \\
(1) & (p \rightarrow q) \rightarrow p, p \rightarrow q & \vdash^? & q; \emptyset & (\textit{reduction}) \\
 & (p \rightarrow q) \rightarrow p & \vdash^? & p;\ (q) & (\textit{reduction}) \\
 & & \vdash^? & p \rightarrow q;\ (q, p) & (\textit{implication}) \\
(2) & p & \vdash^? & q;\ (q, p) & (\textit{bounded restart}) \\
 & p & \vdash^? & p;\ (q, p, q) & (\textit{success})
\end{array}$$

A goal-directed calculus for the implicational fragment of *classical logic* is obtained by liberalising (*bounded restart*) to allow restarts from *any* previous goal [5].

Definition 3.3 (**GDLK**$^{\rightarrow}$)
GDLK$^{\rightarrow}$ has the same rules as **GDLJ**$^{\rightarrow}$ with (*bounded restart*) replaced by:

(*restart*) From $\Gamma \vdash^? q, H$ step to $\Gamma \vdash^? p, H * (q)$ if p occurs in H

Example 3.4
Peirce's axiom (which is not valid in intuitionistic logic) is derivable in **GDLK**$^{\rightarrow}$ as follows, using (*restart*) to continue the deduction at (1):

$$\begin{array}{llcll}
 & & \vdash^? & ((p \rightarrow q) \rightarrow p) \rightarrow p, \emptyset & (\textit{implication}) \\
 & (p \rightarrow q) \rightarrow p & \vdash^? & p, \emptyset & (\textit{reduction}) \\
 & & \vdash^? & p \rightarrow q, (p) & (\textit{implication}) \\
(1) & p & \vdash^? & q, (p) & (\textit{restart}) \\
 & p & \vdash^? & p, (p, q) & (\textit{success})
\end{array}$$

Goal-directed calculi for *Gödel logics* can also be defined by modifying the history to allow states of the database to be recorded, see [17] for details.

[6] In fact the calculus is an $O(n \log n)$-space decision procedure for this fragment [6].

The goal-directed approach is closely related to the Uniform Proof paradigm of Miller and others, see e.g. [21, 7, 10, 11], which has been used as the basis for logic programming in various non-classical logics. A uniform proof of a sequent $\Gamma \vdash A$ is a sequent calculus proof where (reading upwards) the goal A is decomposed first and the rules are applied to formulae in Γ only when A is atomic. In this respect the connectives occurring in the goal A can be seen as "instructions" for directing proof search. A uniform proof calculus for a logic is obtained from an analysis of the permutability of rules of a suitable sequent calculus for that logic. This analysis allows the identification of fragments of the logic, such that if a sequent expressed in this fragment has a proof, then it has a uniform proof. The approach has been applied most successfully to logics (or fragments of logics) having a single-conclusion sequent calculus, although the treatment of multiple-conclusion calculi is also possible. One suitable (propositional) fragment is given by the so-called Harrop-formulae, which intuitively, are formulae that do not contain negative occurrences of "disjunctions". This fragment can be characterized by distinguishing between database (D) and goal (G) formulae, defined by mutual induction as follows:

$$\begin{array}{l} G = \bot \mid \top \mid p \mid G \wedge G \mid G \vee G \mid CD \to G \\ D = \bot \mid p \mid G \to p \\ CD = D \mid CD \mid CD \end{array}$$

In intuitionistic logic every Harrop formula is equivalent to a G-formula. Uniform proof systems have also been defined for fragments of linear logic [11], where formulae in the database may be understood as representing a state that is modified, or resources that are consumed, during the deduction process. The connectives of linear logic give a finer control of the deduction process than intuitionistic connectives, e.g. the multiplicative conjunction of goals prescribes a partition of the available resources.

4 A Basic Goal-Directed System

In this section we define a basic goal-directed system for **Ł** based on the language with connectives $\to$ and $\overline{0}$, recalling that other standard connectives are definable from this pair. We begin by generalizing the queries for intuitionistic and classical logic given in the previous section. A *goal-directed query* for **Ł** consists of a database together with a *multiset* of goals (rather than just one), and a history of previous *states* of the database with goals (rather than just previous goals). More formally:

Definition 4.1 (Goal-Directed Query)
A *goal-directed query* for **Ł** (query) is a structure $\Gamma \vdash^? \Delta; H$ where:

1. Γ is a multiset of formulae called the *database*.
2. Δ is a multiset of formulae called the *goals*.
3. H is a multiset of pairs of multisets of formulae (states of the database with goals), written $\{(\Gamma_1 \vdash^? \Delta_1), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$, called the *history*.

A query is *atomic* if all formulae occurring in it are atomic, i.e. constants or propositional variables. The *size* of a query is the number of symbols occurring in it.

Note that instead of sets or sequences, we consider *multisets* of formulae. Hence the multiplicity but not the order of elements is important. Note also that we write nA

for the multiset containing n copies of a formula A, $n\Gamma$ for the multiset union of n copies of the multiset Γ, and $|\Gamma|$ for the number of elements in Γ.

A query may be understood intuitively as asking if either the goals "follow from" the database or there is an alternative state in the history where this holds. Formally we define the validity of queries as follows:

Definition 4.2 (Validity of Queries)
A query $Q = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2); \ldots ; (\Gamma_n \vdash^? \Delta_n)\}$ is *valid* in $\mathbf{Ł}$, written $\models^*_{\mathbf{Ł}} Q$, iff for all valuations v for $\mathbf{Ł}$,

$$\#^v_{\mathbf{Ł}}\Gamma_i \leq \#^v_{\mathbf{Ł}}\Delta_i \ \text{ for some } i,\ 1 \leq i \leq n,$$

where $\#^v_{\mathbf{Ł}}(\Gamma) = 1 + \sum_{A\in\Gamma}\{v(A) - 1\}$ for a multiset of formulae Γ.

We emphasize that for formulae (i.e. a goal with an empty database and history) our interpretation agrees with the usual one for $\mathbf{Ł}$.

Lemma 4.3
For a formula A, $\models^*_{\mathbf{Ł}} \vdash^? A; \emptyset$ iff $\models_{\mathbf{Ł}} A$.

Proof. Direct from Definition 4.2. ■

Observe that a query is valid if every goal A matches with an occurrence of either A or $\overline{0}$ in the database. We express this fact using the following relation:

Definition 4.4 ($\subseteq^*$)
For multisets of formulae Γ and Δ, $\Delta \subseteq^* \Gamma$ iff:

1. $\Gamma = \Gamma_1 \cup n\overline{0}$.
2. $\Delta = \Delta_1 \cup \Delta_2$.
3. $\Delta_1 \subseteq \Gamma_1$.
4. $|\Delta_2| \leq n$.

Lemma 4.5
If $\Delta \subseteq^* \Gamma$ then $\models^*_{\mathbf{Ł}} \Gamma \vdash^? \Delta; H$.

Proof. A simple induction on $|\Delta|$. ■

We now define a basic goal-directed calculus for $\mathbf{Ł}$.

Definition 4.6 (**GDŁ**)
GDŁ has the following rules:

(suc)	$\Gamma \vdash^? \Delta; H$ succeeds if $\Delta \subseteq^* \Gamma$
(imp)	From $\Gamma \vdash^? \Pi \to q, \Delta; H$ step to $\Gamma \vdash^? \Delta; H$ and $\Gamma, \Pi \vdash^? q, \Delta; H$
(red_l)	From $\Gamma, \Pi \to q \vdash^? q, \Delta; H$ step to $\Gamma \vdash^? \Pi, \Delta; H \cup \{(\Gamma \vdash^? q, \Delta)\}$
(red_r)	From $\Gamma \vdash^? q, \Delta; H \cup \{(\Gamma', \Pi \to q \vdash^? \Delta')\}$ step to: $\Gamma', q \vdash^? \Pi, \Delta'; H \cup \{(\Gamma' \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$
(red_n)	From $\Gamma, \Pi \to \overline{0} \vdash^? \Delta; H$ step to $\Gamma, \overline{0} \vdash^? \Pi, \Delta; H \cup \{(\Gamma \vdash^? \Delta)\}$
(mgl)	From $\Gamma \vdash^? q, \Delta; H \cup \{(\Gamma', q \vdash^? \Delta')\}$ step to: $\Gamma, \Gamma' \vdash^? \Delta, \Delta'; H \cup \{(\Gamma', q \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$
(rst)	From $\Gamma \vdash^? \Delta; H \cup \{(\Gamma' \vdash^? \Delta')\}$ step to $\Gamma' \vdash^? \Delta'; H \cup \{(\Gamma \vdash^? \Delta)\}$

Note that the implication rule (*imp*) treats a query with an implicational goal $\Pi \to q$, and steps to *two* further queries: one where this goal is removed, and one where Π is added to the database and q replaces $\Pi \to q$ as a goal. We can also define a version where $\Pi \to q$ is the only goal in which case only one premise is needed, i.e.:

$$(imp') \text{ From } \Gamma \vdash^? \Pi \to q; H \text{ step to } \Gamma, \Pi \vdash^? q; H$$

Note also that for convenience of presentation we have split the reduction process of database formulae into three "reduction rules". *Local reduction* (red_l) and *remote reduction* (red_r) treat the cases where a goal matches the head of a formula in the database, and in the database of a state in the history, respectively, and *negation reduction* (red_n) decomposes a formula in the database with head $\overline{0}$. Finally, observe that the mingle rule (*mgl*) combines two states in the case where one has a goal occurring in the database of the other, and that the restart rule (*rst*) allows the computation to switch to a state recorded in the history.

GDŁ can be adapted to provide goal-directed calculi for other logics. For example a calculus for *classical logic* is obtained by changing all mention in Definition 4.1 of multisets to sets, and replacing the restart rule with:

$$(res') \text{ From } \Gamma \vdash^? \Delta; H \cup \{(\Gamma' \vdash^? \Delta')\} \text{ step to } \Gamma', \Gamma \vdash^? \Delta'; H \cup \{(\Gamma \vdash^? \Delta)\}$$

Example 4.7
We illustrate **GDŁ** with a proof of the axiom (Ł4), using (1) and (2) to mark separate branches of the derivation:

$$\begin{array}{rrcll}
 & & \vdash^? & [(p \to q) \to q, q \to p] \to p; \emptyset & (imp') \\
 & (p \to q) \to q, q \to p & \vdash^? & p; \emptyset & (red_l) \\
 & (p \to q) \to q & \vdash^? & q; \{((p \to q) \to q \vdash^? p)\} & (red_l) \\
 & & \vdash^? & p \to q; \{(\vdash^? q), ((p \to q) \to q \vdash^? p)\} & (imp') \\
 & p & \vdash^? & q; \{(\vdash^? q), ((p \to q) \to q \vdash^? p)\} & (red_r) \\
 & q & \vdash^? & p, p \to q; \{(\vdash^? q), (p \vdash^? q)\} & (imp) \\
(1) & q & \vdash^? & p; \{(\vdash^? q), (p \vdash^? q)\} & (mgl) \\
 & q & \vdash^? & q; \{(\vdash^? q), (p \vdash^? q), (q \vdash^? p)\} & (suc) \\
(2) & q, p & \vdash^? & p, q; \{(\vdash^? q), (p \vdash^? q), (q \vdash^? p)\} & (suc)
\end{array}$$

GDŁ is *sound* with respect to the interpretation of queries given in Definition 4.2.

Theorem 4.8 (Soundness of **GDŁ**)
If a query Q succeeds in **GDŁ**, then $\models^*_{\mathbf{Ł}} Q$.

Proof. We proceed by induction on the height h of a derivation of Q in **GDŁ**. If $h = 0$ then Q succeeds by (*suc*) and we are done by Lemma 4.5. For $h > 0$ we have the following cases, noting that if the history H is repeated in both premises and conclusion of a rule, then we can assume that $H = \emptyset$:

- (*imp*). Consider a valuation v. If $\#^v_{\mathbf{Ł}}\Pi \leq v(q)$, then $v(\Pi \to q) = 1$ and, since, by the first premise $\#^v_{\mathbf{Ł}}\Gamma \leq \#^v_{\mathbf{Ł}}\Delta$, we are done. Otherwise, $v(\Pi \to q) = 1 - \#^v_{\mathbf{Ł}}\Pi + v(q)$, and we are also done, since, by the second premise, $\#^v_{\mathbf{Ł}}\Gamma + \#^v_{\mathbf{Ł}}\Pi - 1 \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$.

- (red_l), (red_r), (red_n). We just consider (red_l), the other two being very similar. Given a valuation v for **Ł**, either $\#^v_{\mathbf{Ł}}\Gamma \leq \#^v_{\mathbf{Ł}}\Pi + \#^v_{\mathbf{Ł}}\Delta - 1$ or $\#^v_{\mathbf{Ł}}\Gamma \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$. In either case, since $v(\Pi \to q) \leq 1$, and $v(\Pi \to q) \leq v(q) - \#^v_{\mathbf{Ł}}\Pi + 1$, we get that $\#^v_{\mathbf{Ł}}\Gamma + v(\Pi \to q) - 1 \leq \#^v_{\mathbf{Ł}}\Delta$ as required.
- (mgl). Given a valuation v for **Ł**, if either $\#^v_{\mathbf{Ł}}\Gamma' + v(q) - 1 \leq \#^v_{\mathbf{Ł}}\Delta'$ or $\#^v_{\mathbf{Ł}}\Gamma \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$, then we are done, so assume that $\#^v_{\mathbf{Ł}}\Gamma + \#^v_{\mathbf{Ł}}\Gamma' - 1 \leq \#^v_{\mathbf{Ł}}\Delta + \#^v_{\mathbf{Ł}}\Delta' - 1$. Now if $\#^v_{\mathbf{Ł}}\Gamma' + v(q) - 1 > \#^v_{\mathbf{Ł}}\Delta'$, then, by simple arithmetic, $\#^v_{\mathbf{Ł}}\Gamma \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$ and we are done.
- (rst). Trivial since the premise and conclusion have the same interpretation. ∎

To prove *completeness* we distinguish two functions of **GDŁ**: (1) to decompose complex formulae using rules that step from valid queries to valid queries, and (2) to determine the validity of atomic queries. We show that (1) allows us to reduce the derivability of a query to the derivability of queries containing only atoms and irrelevant (in the sense that they do not affect the validity of the query) formulae, and that (2) allows us to derive all valid atomic queries. We begin with the former.

Definition 4.9 (Invertible)
A rule is *invertible* if whenever its conclusion is valid, then all its premises are valid.

Lemma 4.10
(red_l), (red_r), (red_n), (imp), and (rst) are invertible.

Proof. We consider each case in turn:

- (imp). Suppose that, for a valuation v, $\#^v_{\mathbf{Ł}}\Gamma \leq \#^v_{\mathbf{Ł}}\Delta + v(\Pi \to q) - 1$. The result follows for both premises by observing that $v(\Pi \to q) \leq 1$ and $v(\Pi \to q) \leq v(q) - \#^v_{\mathbf{Ł}}\Pi + 1$.
- (red_l), (red_r), (red_n). Again we just consider (red_l). Given a valuation v, if $\#^v_{\mathbf{Ł}}\Pi \leq v(q)$, then $v(\Pi \to q) = 1$. From the conclusion $\#^v_{\mathbf{Ł}}\Gamma \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$ and we are done since $\Gamma \vdash^? q, \Delta$ occurs in the new history. If $\#^v_{\mathbf{Ł}}\Pi > v(q)$ then $v(\Pi \to q) = v(q) - \#^v_{\mathbf{Ł}}\Pi + 1$ and from the conclusion $\#^v_{\mathbf{Ł}}\Gamma + \#^v_{\mathbf{Ł}}\Pi - 1 \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$, and we are done.
- (rst). Trivial since the premise and conclusion have the same interpretation. ∎

If we apply (red_l), (red_r), (red_n), and (imp) exhaustively to a query using (rst) to move between different states of the history, then we end up with queries where the states are atomic goals that fails to match the head of a non-atomic formula in the database of *any* state. We call such queries *irreducible.*

Definition 4.11 (Irreducible)
A query $\Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ is *irreducible* iff:

1. Δ_i is atomic for $i = 1, \ldots, n$.
2. $\Gamma_i = \Pi_i \cup \Sigma_i$ for $i = 1, \ldots, n$ where Σ_i is atomic.
3. $Head(\Pi_1 \cup \ldots \cup \Pi_n) \cap (\Delta_1 \cup \ldots \cup \Delta_n \cup \{\overline{0}\}) = \emptyset$.

Lemma 4.12
For a valid query Q we can apply the rules of **GDŁ** to step to valid irreducible queries.

Proof. We define the following measures and well-orderings:

$c(q) = 1$ for q atomic, $c(A \rightarrow B) = c(A) + c(B) + 1$ for formulae A, B.
$mc(\Gamma \vdash^? \Delta) = \{c(A) \mid A \in \Gamma \cup \Delta\}$ for multisets Γ, Δ.
$mmc(\Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}) = \{mc(\Gamma_i \vdash^? \Delta_i) \mid 1 \leq i \leq n\}$.

For multisets α, β of integers: $\alpha <_m \beta$ iff (1) $\alpha \subset \beta$, or (2) $\alpha <_m \gamma$ where $\gamma = (\beta - \{j\}) \cup \{i, \ldots, i\}$, and $i < j$.

For multisets ϕ, ψ of multisets of integers, $\phi <_{mm} \psi$ iff (1) $\phi \subset \psi$, or (2) $\phi <_{mm} \chi$ where $\chi = (\psi - \{\alpha\}) \cup \{\beta, \ldots, \beta\}$ and $\beta <_m \alpha$.

We apply (red_l), (red_r), (red_n), and (imp) exhaustively to a query using (rst) to move between different states of the history. This process terminates since for each rule where Q steps to Q', $mmc(Q') < mmc(Q)$. Hence we arrive at queries which, since (imp) cannot be applied, must have atomic goals, and, since (red_l), (red_r), and (red_n) cannot be applied, do not have a database formulae whose head matches a goal or $\overline{0}$. ∎

If an irreducible query is valid, then by removing non-atomic formulae we obtain an atomic query that is also valid.

Lemma 4.13
If the following conditions hold:

1. $\models^*_{\text{Ł}} \Gamma_1, \Pi_1 \vdash^? \Delta_1; \{(\Gamma_2, \Pi_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Pi_n \vdash^? \Delta_n)\}$
2. Γ_i and Δ_i are atomic for $i = 1, \ldots, n$
3. $Head(\Pi_1 \cup \ldots \cup \Pi_n) \cap (\Delta_1 \cup \ldots \cup \Delta_n \cup \{\overline{0}\}) = \emptyset$

Then: $\models^*_{\text{Ł}} \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$

Proof. Assume $\not\models^*_{\text{Ł}} \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$. We get that there is a valuation v for **Ł** such that $\#^v_{\text{Ł}}\Gamma_i > \#^v_{\text{Ł}}\Delta_i$ for $i = 1, \ldots, n$. We define a new valuation w as follows:

$$w(q) \quad = \quad \begin{cases} 1 & \text{if } q \in Head(\Pi_1 \cup \ldots \cup \Pi_n) \\ v(q) & \text{otherwise} \end{cases}$$

Since $Head(\Pi_1 \cup \ldots \cup \Pi_n) \cap (\Delta_1 \cup \ldots \cup \Delta_n) = \emptyset$ we have that $\#^w_{\text{Ł}}\Delta_i = \#^v_{\text{Ł}}\Delta_i$ for $i = 1, \ldots, n$. We also get that $\#^w_{\text{Ł}}\Pi_i = 1$ and $\#^w_{\text{Ł}}\Gamma_i \geq \#^v_{\text{Ł}}\Gamma_i$ for $i = 1, \ldots, n$. Hence $\#^w_{\text{Ł}}\Pi_i + \#^w_{\text{Ł}}\Gamma_i - 1 > \#^w_{\text{Ł}}\Delta_i$ for $i = 1, \ldots, n$, i.e. $\not\models^*_{\text{Ł}} \Gamma_1, \Pi_1 \vdash^? \Delta_1; \{(\Gamma_2, \Pi_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Pi_n \vdash^? \Delta_n)\}$, a contradiction. ∎

The next step is to show that all valid atomic (and hence also irreducible) queries succeed in **GDŁ**. To this end we introduce a translation from irreducible queries containing propositional variables $q_1, \ldots, q_n$ into a set of inequations in the variables $x_{q_1}, \ldots, x_{q_n}$ over $[-1, 0]$.

Definition 4.14 ($\mathbf{S_Q}$)
Let $Q = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ where Δ_i is atomic for $i = 1, \ldots, n$:

$$S_Q = \{ \sum_{q \in \Gamma_i^a} x_q > \sum_{q \in \Delta_i} x_q \; : \; 1 \leq i \leq n\}$$

where $\Gamma^a = \{A \in \Gamma \ : \ A \text{ is atomic}\}$, x_q is a real-valued variable for all propositional variables q, and $x_{\overline{0}} = -1$.

Lemma 4.15
Let Q be an atomic query. If $\models^*_{\mathbf{Ł}} Q$ then Q succeeds in **GDŁ**.

Proof. Let $Q = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$. If $\models_{\mathbf{Ł}} Q$ then for all valuations v for **Ł**, $\#^v_{\mathbf{Ł}}\Gamma_i \leq \#^v_{\mathbf{Ł}}\Delta_i$ for some i, $1 \leq i \leq n$. Equivalently, the set S_Q is inconsistent over $[-1, 0]$. By linear programming methods (see e.g. [29] for details) there exist $\lambda_1, \ldots, \lambda_n \in \mathbb{N}$ such that $\lambda_i > 0$ for some $1 \leq i \leq n$ and:

$$\bigcup_{i=1}^{n} \lambda_i \Delta_i \subseteq^* \bigcup_{i=1}^{n} \lambda_i \Gamma_i$$

We show that Q succeeds in **GDŁ** by induction on $\lambda = \sum_{i=1}^n \lambda_i$. If $\lambda = 1$ then Q succeeds by an application of (rst) if necessary and (suc). For $\lambda > 1$ we have two cases. First suppose that for some $i \neq j$, $1 \leq i, j \leq n$, we have $\lambda_i, \lambda_j > 0$ and there exists $q \in \Delta_i \cap \Gamma_j$. Since we can always apply (rst), we can assume without loss of generality that $i = 1$ and $j = 2$. Now by applying (mgl) we obtain a query:

$$Q' = \Gamma_1, \Gamma_2 - \{q\} \vdash^? \Delta_1 - \{q\}, \Delta_2; \{(\Gamma_1 \vdash^? \Delta_1), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$$

If $\lambda_1 \geq \lambda_2$ then we have:

$$(\lambda_1 - \lambda_2)\Delta_1 \cup \lambda_2(\Delta_1 - \{q\} \cup \Delta_2) \cup \bigcup_{i=3}^{n} \lambda_i \Delta_i \subseteq^* (\lambda_1 - \lambda_2)\Gamma_1 \cup \lambda_2(\Gamma_1 \cup \Gamma_2 - \{q\}) \cup \bigcup_{i=3}^{n} \lambda_i \Gamma_i$$

Moreover, $(\lambda_1 - \lambda_2) + \lambda_2 + \sum_{i=3}^n \lambda_i < \lambda$, and hence, by the induction hypothesis, Q' succeeds in **GDŁ** so we are done. The case where $\lambda_2 \geq \lambda_1$ is very similar. Alternatively, suppose that for all $q \in \Delta_i$ for $i = 1, \ldots, n$, where $\lambda_i > 0$, there is no j such that $\lambda_j > 0$ and $q \in \Gamma_j$. In at least one state there must be more occurrences of $\overline{0}$ than goals, so the query succeeds by (rst) and (suc). ∎

Lemma 4.16
If $Q = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ succeeds in **GDŁ** then $Q' = \Gamma_1, \Pi_1 \vdash^? \Delta_1; \{(\Gamma_2, \Pi_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Pi_n \vdash^? \Delta_n)\}$ succeeds in **GDŁ**.

Proof. A simple inductive proof on the length of a derivation of Q. ∎

Lemma 4.17
Let Q be an irreducible query. If $\models^*_{\mathbf{Ł}} Q$ then Q succeeds in **GDŁ**.

Proof. By definition $Q = \Gamma_1, \Pi_1 \vdash^? \Delta_1; \{(\Gamma_2, \Pi_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Pi_n \vdash^? \Delta_n)\}$ where Γ_i and Δ_i are atomic for $i = 1, \ldots, n$. If $\models^*_{\mathbf{Ł}} Q$, then, by Lemma 4.13, also $\models^*_{\mathbf{Ł}} Q'$ where $Q' = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$. Q' succeeds by Lemma 4.15, so, by Lemma 4.16, Q also succeeds. ∎

Theorem 4.18 (Completeness of **GDŁ**)
For a query Q, if $\models^*_{\mathbf{Ł}} Q$ then Q succeeds in **GDŁ**.

Proof. We apply the invertible rules to Q, terminating with valid irreducible queries by Lemmas 4.10 and 4.12. These succeed by Lemma 4.17. ∎

5 Termination and Efficiency

In this section we make a number of refinements to **GDŁ**, both to improve efficiency, and to ensure termination. First, by introducing new propositional variables into the reduction rules, we prevent the duplication of formulae, avoiding an exponential growth in the size of queries. We then consider termination, introducing a revised success rule that transforms irreducible queries into linear programming problems. We show that, together with the modification to the reduction rules, this revised calculus provides the basis for a co-NP algorithm for checking validity in **Ł**. As an alternative we also give revised mingle rules that provide a purely logical basis for obtaining termination.

5.1 More efficient reduction

One problem compromising the efficiency of **GDŁ** is that the reduction rules place multiple copies of the database in the history. As a consequence, each formula occurring in a query may end up being reduced an exponential number of times. For example, notice that in Example 4.7 the formula $(p \to q) \to q$ occurs (3 lines down) twice: once in the database, and once again in the history. The solution we propose here is to introduce *new propositional variables* during the reduction process. These ensure that formulae are not duplicated, and may be thought of as markers keeping track of different options in the history. For the sake of clarity we use x, y, z to denote these new variables. However, we emphasize that these are treated in exactly the same way as all other variables occurring in a query. Hence our queries still have a *logical interpretation*.

Definition 5.1 (Revised reduction rules)
We define the following *revised reduction rules* (recalling that $2x$ stands for $\{x, x\}$):

(red_l) From $\Gamma, \Pi \to q' \vdash^? q, \Delta; H$ where $q' \in \{q, \overline{0}\}$ step to:
$q' \vdash^? \Pi, 2x; H \cup \{(\Gamma, 2x \vdash^? q, \Delta)\}$ where x is new.

(red_r) From $\Gamma \vdash^? q, \Delta; H \cup \{\Gamma', \Pi \to q \vdash^? \Delta'\}$ step to:
$q \vdash^? \Pi, 2x; H \cup \{(\Gamma', 2x \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$ where x is new.

These rules give a different breakdown of the reduction process. (red_l) takes care of the cases where the head of a formula in the database is $\overline{0}$ or matches a goal. (red_r) takes care of the cases where a goal matches the head of a formula in a state of the database in the history. In both (red_l) and (red_r), the formula is replaced by two copies of a new propositional variable x, which act as markers allowing the formula to be decomposed independently from the current database. The fact that *two* copies of the variable are used is due to the interpretation of x in the interval $[0, 1]$, since, for soundness, we may require a value outside this interval.

Example 5.2
We illustrate the revised reduction rules with the following proof:

$$\begin{array}{rcll}
 & \vdash^? & [(p \to q) \to r, (q \to p) \to r] \to r; \emptyset & (imp') \\
(p \to q) \to r, & & & \\
(q \to p) \to r & \vdash^? & r; \emptyset & (red_l) \\
r & \vdash^? & p \to q, 2x; \{(2x, (q \to p) \to r \vdash^? r)\} & (imp) \\
(1) \quad r & \vdash^? & 2x, \{(2x, (q \to p) \to r \vdash^? r)\} & (mgl) \\
r, x, (q \to p) \to r & \vdash^? & x, r, \{(r \vdash^? 2x), (2x, (q \to p) \to r \vdash^? r)\} & (suc) \\
(2) \quad r, p & \vdash^? & q, 2x; \{(2x, (q \to p) \to r \vdash^? r)\} & (rst) \\
2x, (q \to p) \to r & \vdash^? & r; \{(r, p \vdash^? q, 2x)\} & (red_l) \\
r & \vdash^? & q \to p, 2y; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r)\} & (imp) \\
(2.1) \quad r & \vdash^? & 2y; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r)\} & (mgl) \\
2x, r, y & \vdash^? & r, y; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r), (r \vdash^? 2y)\} & (suc) \\
(2.2) \quad r, q & \vdash^? & p, 2y; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r)\} & (mgl) \\
2r, q & \vdash^? & q, 2x, 2y; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r), & \\
 & & \qquad (r, q \vdash^? p, 2y)\} & (mgl) \\
2r, q, x, 2y & \vdash^? & q, x, 2y, r; \{(r, p \vdash^? q, 2x), (2x, 2y \vdash^? r), & \\
 & & \qquad (r, q \vdash^? p, 2y), (2r, q \vdash^? q, 2x, 2y)\} & (suc)
\end{array}$$

Lemma 5.3
For a query Q, applying the revised reduction rules, (imp), and (rst), gives irreducible queries of size polynomial in the size of Q.

Proof. Let n be the number of occurrences of $\to$ in Q. Each application of (red_l) and (red_r) increases the size of the query at most linearly, and gives one fewer occurrences of $\to$. Hence there can be at most n such applications, and the size of the resulting irreducible queries must be polynomial in the size of Q. ∎

Theorem 5.4
For a query Q, $\models^*_{\mathbf{Ł}} Q$ iff Q succeeds in **GDŁ** with the revised reduction rules.

Proof. For soundness, it is sufficient to check the soundness of the revised reduction rules, assuming as before that $H = \emptyset$. Consider (red_l), the case of (red_r) being very similar. Suppose that we have a valuation v for **Ł**. If $\#^v_{\mathbf{Ł}}\Pi < v(q)$ then $v(\Pi \to q) = 1$ and, extending v with $v(x) = 1$, we get from the premise that $\#^v_{\mathbf{Ł}}\Gamma = \#^v_{\mathbf{Ł}}\Gamma + 2v(x) - 2 \leq v(q) + \#^v_{\mathbf{Ł}}\Delta$ as required. If $\#^v_{\mathbf{Ł}}\Pi \geq v(q)$ then suppose that $\#^v_{\mathbf{Ł}}\Gamma + v(\Pi \to q) - 1 = \#^v_{\mathbf{Ł}}\Gamma + v(q) - \#^v_{\mathbf{Ł}}\Pi - 2 > v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$. This means that $\#^v_{\mathbf{Ł}}\Gamma - \#^v_{\mathbf{Ł}}\Pi - 1 + \epsilon > \#^v_{\mathbf{Ł}}\Delta$ for some $\epsilon < 0$. We now define:

$$v(x) = \frac{v(q) - \#^v_{\mathbf{Ł}}\Pi + \epsilon}{2}$$

noting that dividing by two ensures that $0 \leq v(x) \leq 1$, which gives $v(q) > \#^v_{\mathbf{Ł}}\Pi + 2v(x) - 2$ and $\#^v_{\mathbf{Ł}}\Gamma + 2v(x) - 2 > \#^v_{\mathbf{Ł}}\Delta$, a contradiction.

For completeness we proceed in exactly the same way as for **GDŁ**. Hence it is enough to show that the revised reduction rules are invertible. We just treat the case of (red_l), (red_r) being very similar. Suppose that we have a valuation v for **Ł**. If $\#^v_{\mathbf{Ł}}\Pi \leq v(q')$, then $v(\Pi \to q') = 1$ and so, from the validity of the conclusion, we get $\#^v_{\mathbf{Ł}}\Gamma + 2v(x) - 2 \leq \#^v_{\mathbf{Ł}}\Gamma \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$ as required. If $\#^v_{\mathbf{Ł}}\Pi \geq v(q)$, then either $v(x) \geq v(q) - \#^v_{\mathbf{Ł}}\Pi$ in which case we are done, or $v(x) \leq v(q) - \#^v_{\mathbf{Ł}}\Pi$ and we get $\#^v_{\mathbf{Ł}}\Gamma + v(x) \leq v(q) + \#^v_{\mathbf{Ł}}\Delta - 1$ as required. ■

5.2 Terminating calculi

There are several options for obtaining terminating calculi for **Ł**, the simplest being to feed queries to a linear programming problem solver using Definition 4.14, i.e.

Definition 5.5 (Revised Success Rule)
We define the following *revised success rule*:

$$(suc) \quad Q = \Gamma_1 \vdash^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\} \text{ succeeds if}$$
$$\Delta_i \text{ is atomic for } i = 1, \ldots, n \text{ and } S_Q \text{ is inconsistent over } [-1, 0].$$

Recall that validity of an irreducible query Q is equivalent to the inconsistency of S_Q.

Example 5.6
$Q = p \to r, q, \overline{0} \vdash^? p, p; \{(q \to r, p \vdash^? q)\}$ is transformed into the set of inequations:

$$S_Q = \{x_q - 1 > 2x_p, p > q\}$$

which is inconsistent over $[-1, 0]$.

Theorem 5.7
GDŁ with the revised success and reduction rules gives a co-NP decision procedure for **Ł**.

Proof. To show that a formula A is not valid we apply the revised reduction rules, (imp), and (rst), exhaustively to $Q = \vdash^? A; \emptyset$, making a non-deterministic choice of two branches where necessary. By Lemma 5.3, we arrive at an irreducible query Q' of size polynomial in the size of Q, where A is valid iff Q' is valid. Q' is valid iff $S^{Q'}$ is inconsistent, and this can be checked in polynomial time using linear programming. ■

We can also easily cope with *finite-valued* Łukasiewicz logics. For the k-valued logic $\mathbf{Ł_k}$ we just change the condition in the (suc) rule to:

$$S_Q \text{ is inconsistent over } [-1, -\tfrac{k-2}{k-1}, \ldots, -\tfrac{1}{k-1}, 0]$$

Alternatively, we may define "logical methods" for solving linear programming problems by changing the (mgl) rule to allow matching of multiple occurrences of an atom e.g. $4p$ with $3p$. A terminating procedure is then obtained by removing all matching occurrences of one particular propositional variable at a time. We give two versions:

Definition 5.8 (Revised Mingle Rules)
We define the following *revised mingle rules*:

(mgl_1) From $\Gamma \vdash^? \lambda q, \Delta; H \cup \{(\Gamma', \mu q \vdash^? \Delta')\}$ step to:
$\mu\Gamma, \lambda\Gamma' \vdash^? \mu\Delta, \lambda\Gamma'; H \cup \{(\Gamma', \mu q \vdash^? \Delta'), (\Gamma \vdash^? \lambda q, \Delta)\}$

(mgl_2) From $\Gamma \vdash^? \lambda q, \Delta; H \cup \{(\Gamma', \mu q \vdash^? \Delta')\}$ step to:
either $\mu\Gamma, \lambda\Gamma' \vdash^? \mu\Delta, \lambda\Gamma'; H \cup \{(\Gamma \vdash^? \lambda q, \Delta)\}$
or $\mu\Gamma, \lambda\Gamma' \vdash^? \mu\Delta, \lambda\Gamma'; H \cup \{(\Gamma', \mu q \vdash^? \Delta'\}$

(mgl_1) keeps all previous states of the database in the history, giving an exponential blow-up in the size of the query. (mgl_2) on the other hand keeps the size of queries linear but requires non-deterministic choices. The reduction rule (red_l) is necessary here to remove atomic goals occurring in the database. We can also define:

(red_l') From $\Gamma, \Sigma \vdash^? \Sigma, \Delta; H$ step to $\Gamma \vdash^? \Delta; H$

Example 5.9
We illustrate (mgl_2) with a proof of an atomic query:

$$\begin{array}{rcll}
2p & \vdash^? & 3q, r; \{(q, r \vdash^? p), (2q \vdash^? p)\} & (mgl_2) \\
2p, 3r & \vdash^? & r, 3p; \{(2q \vdash^? p), (2p \vdash^? 3q, r)\} & (red_l') \\
2r & \vdash^? & p; \{(2q \vdash^? p), (2p \vdash^? 3q, r)\} & (rst) \\
2p & \vdash^? & 3q, r; \{(2q \vdash^? p), (2r \vdash^? p)\} & (mgl_2) \\
4p & \vdash^? & 2r, 3p; \{(2q \vdash^? p), (2r \vdash^? p)\} & (red_l') \\
p & \vdash^? & 2r; \{(2q \vdash^? p), (2r \vdash^? p)\} & (mgl_2) \\
2p & \vdash^? & 2p; \{(2q \vdash^? p), (2r \vdash^? p)\} & (suc)
\end{array}$$

Theorem 5.10
Q succeeds in **GDŁ** with (mgl) replaced by (mgl_i) iff $\models^*_{\text{Ł}} Q$ for $i = 1, 2$.

Proof. Very similar to previous proofs. ∎

Theorem 5.11
GDŁ with (mgl_i) terminates with a suitable control strategy for $i = 1, 2$.

Proof. We sketch suitable control strategies. The first step is to reduce formulae in the query to obtain irreducible queries. Since the reduction rules all reduce the complexity of the query it is sufficient here to ensure that $(switch)$ is not applied ad infinitum e.g. using a split history to show which states have already been considered. For the second step we require that (red_l) is applied eagerly to atomic formulae in the database i.e. whenever possible, and that (mgl_i) is applied with the maximum matching propositional variables i.e. we add the condition $q \notin \Gamma \cup \Gamma' \cup \Delta \cup \Delta'$. Moreover we must insist that (mgl_i) is applied exhaustively to just one propositional variable at a time, using $(switch)$ to move between states. ∎

6 Towards Fuzzy Logic Programming

In this section we present an extension of **GDŁ** suitable as a basis for fuzzy logic programming. To simplify matters, we remain at the propositional level. Note however, that (since we use a restricted function-free language) first-order clauses for a finite domain can be translated into propositional logic by considering all possible instances. We leave the important question of finding a more efficient approach using unification for future work. Given this simplified framework, we consider the following logic programming problem. Let Γ be a set of database formulae (clauses) with truth value restrictions of the form $\leq r$ or $\geq r$, where $r \in [0,1] \cap \mathbb{Q}$. For a goal formula G, we want to obtain the maximum value $c \in [0,1]$ satisfying: for all valuations v for **Ł**, if for all $A \in \Gamma$, $v(A)$ obeys the associated truth conditions, then $v(G) \geq c$ for all valuations v. This problem can be translated into a logical consequence problem in Rational Pavelka logic **RPL**. First note that in **RPL** for any valuation v for **RPL**, for any formula A and $r \in [0,1] \cap \mathbb{Q}$:

$$v(A) \geq r \text{ iff } v(\overline{r} \to A) = 1 \text{ and } v(A) \leq r \text{ iff } v(A \to \overline{r}) = 1$$

Hence if $\Gamma = \{D_1, \ldots, D_n\}$, with restrictions, e.g. $\geq r_i$ for $1 \leq i \leq n$, we can express the problem as that of determining the greatest c such that

$$\overline{r}_1 \to D_1, \ldots, \overline{r}_n \to D_n \models_{\text{Ł}} \bar{c} \to G$$

We begin below by giving normal forms for database and goal formulae, that, unlike other Łukasiewicz logic programming approaches, restrict neither the efficiency nor the expressivity of the language. We then give an algorithm which by dealing with logical consequence in **RPL**, facilitates fuzzy logic programming. Finally we give some detailed examples.

6.1 Normal forms

Although the standard connectives of **RPL** are all definable in a language with just $\to$ and $\overline{C}$, this treatment is often unsatisfactory, since the definitions may multiply the complexity of formulae exponentially e.g. $A \vee B =_{def} (A \to B) \to B$. Here we give an alternative approach that deals with other standard connectives of **RPL** in an efficient fashion. We define normal forms for goal and database formulae in a language with connectives $\to$, $\odot$, $\wedge$, $\vee$ and $\overline{0}$. We show that every formula A in this language is equivalent to both a goal formula and a strong conjunction of database formulae, with complexity in both cases linear in the complexity of A.

Definition 6.1 (Normal Forms)
Goal-formulae (G-formulae) and *database-formulae* (D-formulae) are defined by mutual induction as follows:

$$G = q|\overline{r}|G \vee G|G \wedge G|CD \to G \qquad CG = G|CG \odot CG \qquad E = q|\overline{r}|E \vee E$$
$$D = E|CG \to E \qquad CD = D|CD \odot CD$$

Lemma 6.2
For a formula A there exists a G-formula G, and CD-formula CD, such that G and CD have complexity linear in the complexity of A, and $A \equiv_{\text{Ł}} G \equiv_{\text{Ł}} CD$.

Proof. We prove both 1 and 2 by mutual induction on $cp(A)$. If A is atomic then clearly A is both a G-formula and a CD-formula of the right complexity. For A non-atomic we have the following cases:

- $A = A_1 \wedge A_2$. By the induction hypothesis there are G-formulae G_1 and G_2 such that $A_i \equiv_{\mathbf{Ł}} G_i$ for $i = 1, 2$. Hence $G_1 \wedge G_2$ is a G-formula logically equivalent to A. There are also CD-formulae CD_1 and CD_2 such that $A_i \equiv_{\mathbf{Ł}} CD_i$ for $i = 1, 2$. Hence $((CD_1 \rightarrow \overline{0}) \vee (CD_2 \rightarrow \overline{0})) \rightarrow \overline{0})$ is a CD-formula logically equivalent to A.
- $A = A_1 \vee A_2$. Very similar to the previous case, except that if A_1 and A_2 are both atomic then A is already a CD-formula.
- $A = A_1 \odot A_2$. By the induction hypothesis there are G-formulae G_1 and G_2 such that $A_i \equiv_{\mathbf{Ł}} G_i$ for $i = 1, 2$. $((G_1 \odot G_2) \rightarrow \overline{0}) \rightarrow \overline{0}$ is a G-formula logically equivalent to A as required. Also by the induction hypothesis there are CD-formulae CD_1 and CD_2 such that $A_i \equiv_{\mathbf{Ł}} CD_i$ for $i = 1, 2$. Hence $CD_1 \odot CD_2$ is a CD-formula logically equivalent to A as required.
- $A = A_1 \rightarrow A_2$. By the induction hypothesis there are G-formulae G_1 and G_2 such that $A_i \equiv_{\mathbf{Ł}} G_i$ for $i = 1, 2$, and CD-formulae CD_1 and CD_2 such that $A_i \equiv_{\mathbf{Ł}} CD_i$ for $i = 1, 2$. Hence we have that $CD_1 \rightarrow G_2$ is a G-formula logically equivalent to A. Also, since $CD_1 \rightarrow \overline{0}$ is a G-formula, $G_1 \rightarrow CD_1 \equiv_{\mathbf{Ł}} (G_1 \odot (CD_1 \rightarrow \overline{0})) \rightarrow \overline{0}$ is a CD-formula logically equivalent to A. ∎

6.2 An algorithm for fuzzy logic programming

To define an algorithm suitable as a basis for fuzzy logic programming, we make two important changes to the goal-directed queries used in **GDŁ**. Firstly, to treat logical consequence, we divide the database into two: a temporary part containing formulae that can only be used once, and a permanent part, where formulae can be used as many times as we wish. Secondly, to allow a value to be computed for a given goal, we allow a variable X to appear in the temporary database, and give explicitly a truth constant to be substituted for that variable. We arrive at the following revised definition of goal-directed queries.

Definition 6.3 (Goal-Directed Query)
A *goal-directed query* for **FLPŁ** (query) is a structure $Q[X] = \Gamma | \Sigma \vdash_c^? \Delta; H$ where:

1. $c \in [0, 1] \cap \mathbb{Q}$ and X is a variable.
2. Γ is a multiset of D-formulae and copies of X called the *temporary database*.
3. Σ is a multiset of D-formulae called the *permanent database*.
4. Δ is a multiset of G-formulae called the *goals*.
5. H is a multiset of states of the temporary database with goals, written $\{(\Gamma_1 \vdash^? \Delta_1), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$, called the *history*.

Definition 6.4 (Validity of Queries)
A query $Q = \Gamma_1 | \Sigma \vdash_c^? \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ is *valid* in **RPL**, written $\models_{\mathbf{Ł}}^* Q$, iff for all valuations v for **RPL** such that $v(A) = 1$ for all $A \in \Sigma$,

$$\#_{\mathbf{Ł}}^v \Gamma_i[X/c] \leq \#_{\mathbf{Ł}}^v \Delta_i \text{ for some } i,\ 1 \leq i \leq n$$

Recall that for a fuzzy logic programming problem, we want to calculate the maximal value c such that for a given goal G and database Σ, $c \rightarrow G$ is a logical consequence of Σ. Note that this can be expressed as a query with temporary database containing just X, permanent database Σ, an empty history, and one goal G.

Lemma 6.5
$\Sigma \models_{\text{Ł}} c \to G$ iff $\models^*_{\text{Ł}} X|\Sigma \vdash^?_c G; \emptyset$.

Proof. Direct from Definition 6.4. ∎

To deal with rational constants and logical consequence we give the following revised relation for success.

Definition 6.6 ($\subseteq^*_\Sigma$)
For multisets of formulae Γ and Δ, we define $\Delta \subseteq^*_\Sigma \Gamma$ iff:

1. $\Gamma = \Gamma_1 \cup \{\overline{r}_1, \ldots, \overline{r}_n\}$.
2. $\Delta = \Delta_1 \cup \Delta_2 \cup \{\overline{r}'_1, \ldots, \overline{r}'_m\}$.
3. $\Delta_1 \subseteq \Gamma_1 \cup \lambda\Sigma$ for some $\lambda \in \mathbb{N}$.
4. $|\Delta_2| + \sum_{i=1}^{n}(r_i - 1) \le \sum_{i=1}^{m}(r'_i - 1)$.

Lemma 6.7
Let $Q = \Gamma|\Sigma \vdash^? \Delta; H$ be a query. If $\Delta \subseteq^*_\Sigma \Gamma$ then $\models^*_{\text{Ł}} Q$.

Proof. A simple induction on $|\Delta|$. ∎

We now introduce our fuzzy logic programming calculus **FLPŁ**.

Definition 6.8 (**FLPŁ**)
FLPŁ has the following rules:

(suc) $\Gamma|\Sigma \vdash^?_c \Delta; H$ if $c = max\{r \in [0,1] \ : \ \Delta \subseteq^*_\Sigma \Gamma[X/\overline{r}]\}$

(imp) From $\Gamma|\Sigma \vdash^?_c \Pi \to G, \Delta; H$ step to
$\Gamma|\Sigma \vdash^?_{c_1} \Delta; H$ and $\Gamma, \Pi|\Sigma \vdash^?_{c_2} G, \Delta; H$ where $c = min(c_1, c_2)$

(or) From $\Gamma|\Sigma \vdash^?_c G_1 \vee G_2, \Delta; H$ step to $\Gamma|\Sigma \vdash^?_c G_1, \Delta; H \cup \{(\Gamma \vdash^? G_2, \Delta)\}$

(and) From $\Gamma|\Sigma \vdash^?_c G_1 \wedge G_2, \Delta; H$ step to
$\Gamma|\Sigma \vdash^?_{c_1} G_1, \Delta; H$ and $\Gamma|\Sigma \vdash^?_{c_2} G_2, \Delta; H$ where $c = min(c_1, c_2)$

(red_l) From $\Gamma|\Sigma \vdash^?_c q, \Delta; H$ where $\Pi \to (E \vee q') \in \Gamma \cup \Sigma$ and $q' \in \{q\} \cup \overline{C}$ step to
$\Gamma', \Pi \to E|\Sigma \vdash^?_{c_1} q, \Delta; H$ and $\Gamma', q'|\Sigma \vdash^?_{c_2} \Pi, q, \Delta; H \cup \{(\Gamma' \vdash^? q, \Delta)\}$
where $\Gamma' = \Gamma - \{\Pi \to (E \vee q')\}$ if $\Pi \to (E \vee q') \in \Gamma$, otherwise $\Gamma' = \Gamma$,
and $c = min(c_1, c_2)$

(red_r) From $\Gamma|\Sigma \vdash^?_c q, \Delta; H \cup \{(\Gamma', \Pi \to (E \vee q) \vdash^? \Delta')\}$ step to:
$\Gamma|\Sigma \vdash^?_{c_1} q, \Delta; H \cup \{(\Gamma', \Pi \to E \vdash^? \Delta')\}$ and
$\Gamma', q|\Sigma \vdash^?_{c_2} \Pi, \Delta'; H \cup \{(\Gamma' \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$ where $c = min(c_1, c_2)$

(mgl) From $\Gamma|\Sigma \vdash^?_c q, \Delta; H \cup \{(\Gamma', q \vdash^?_c \Delta')\}$ step to:
$\Gamma, \Gamma'|\Sigma \vdash^?_c \Delta, \Delta'; H \cup \{(\Gamma', q \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$

(rst) From $\Gamma|\Sigma \vdash^?_c \Delta; H \cup \{(\Gamma' \vdash^? \Delta')\}$ step to $\Gamma'|\Sigma \vdash^?_c \Delta'; H \cup \{(\Gamma \vdash^? \Delta)\}$

FLPŁ is sound with respect to the interpretation of queries given in Definition 6.4.

Lemma 6.9
If $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_{c_1} \Delta; H$ and $c_2 \leq c_1$. then $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_{c_2} \Delta; H$

Proof. Follows from Definition 6.4 and the fact that X occurs only in states of the temporary database. ■

Theorem 6.10 (Soundness of **FLPŁ**)
If a query Q succeeds in **FLPŁ** then $\models^*_{\mathbf{Ł}} Q$.

Proof. We proceed by induction on the height h of a derivation of Q in **FLPŁ**. If $h = 0$ then $c = max\{r \in [0,1] \ : \ \Delta \subseteq^*_{\Sigma} \Gamma[X/\overline{r}]\}$ where $Q = \Gamma|\Sigma \vdash^?_c \Delta; H$ and hence we are done by Lemma 6.7. For $h > 0$ we have the following cases, noting as before that if part of the history H is repeated in both premises and conclusion of a rule, then we can assume that $H = \emptyset$.

- (imp). If $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_{c_1} \Delta; H$ and $\models^*_{\mathbf{Ł}} \Gamma, \Pi|\Sigma \vdash^?_{c_2} G, \Delta; H$, then since $c = min(c_1, c_2)$, by Lemma 6.9, $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_c \Delta; H$ and $\models^*_{\mathbf{Ł}} \Gamma, \Pi|\Sigma \vdash^?_c G, \Delta; H$. We then proceed as in Theorem 4.8.
- (or). Suppose that $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_c G_1, \Delta; H \cup \{(\Gamma \vdash^? G_2, \Delta)\}$. Since $v(G_i) \leq v(G_1 \vee G_2)$ for $i = 1, 2$, we also get $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_c G_1 \vee G_2, \Delta; H$ as required.
- (and). If $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_{c_1} G_1, \Delta; H$ and $\Gamma|\Sigma \vdash^?_{c_2} G_2, \Delta; H$, then since $c = min(c_1, c_2)$, by Lemma 6.9, $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_c G_1, \Delta; H$ and $\Gamma|\Sigma \vdash^?_c G_2, \Delta; H$. Hence $\models^*_{\mathbf{Ł}} \Gamma|\Sigma \vdash^?_{c_1} G_1 \wedge G_2, \Delta; H$ as required.
- (red_l), (red_r). We just consider (red_l), the case of (red_r) being very similar. Consider a valuation v for which $v(A) = 1$ for all $A \in \Sigma$. If $v(q') \leq v(E)$, then the validity of the conclusion follows directly from the first premise, using Lemma 6.9. If $v(q') > v(E)$, then the validity of the conclusion follows from the second premise as in Theorem 4.8, using Lemma 6.9.
- (mgl). Follows exactly as in Theorem 4.8.
- (rst). Trivial. ■

To prove the completeness of **FLPŁ** we proceed in two steps. First we show that the algorithm is complete when the permanent database is *empty*.

Lemma 6.11
(imp), (or), (and), (red_l), (red_r), and (rst) are invertible, replacing c_1 and c_2 with c.

Proof. We proceed case by case, assuming as before that the repeated history is empty.

- (imp). Proceeds in exactly the same way as Lemma 4.10.
- (or). Consider a valuation v. If $v(G_1) \leq v(G_2)$ then $v(G_1 \vee G_2) = v(G_2)$ and the result follows since $\Gamma \vdash^? G_2, \Delta$ is in the history of the premise. If $v(G_2) \leq v(G_1)$ then $v(G_1 \vee G_2) = v(G_1)$ and the result follows since the current goal and temporary database of the premise is $\Gamma \vdash^? G_1, \Delta$.
- (and). Follows immediately since for any valuation v, $v(G_1 \wedge G_2) \leq v(G_i)$ for $i = 1, 2$.
- (red_l), (red_r). We just consider (red_l), the case of (red_r) being very similar. The first premise follows immediately since for any valuation v, $v(\Pi \to E) \leq v(\Pi \to (E \vee q'))$. The second premise follows in exactly the same way as Lemma 4.10.

• (rst). Trivial. ∎

Lemma 6.12
For $Q = \Gamma|\emptyset \vdash^?_c \Delta; H$, if $\models^*_{\mathbf{Ł}} Q$ then Q succeeds in **FLPŁ**.

Proof. Since the permanent database is empty we are able to follow the corresponding proof for **GDŁ**, noting that no rule of **FLPŁ** introduces formulae into the permanent database. Hence, as before, applying the invertible rules (imp), (or), (and), (red_l), and (red_r), using (rst) to move between states of the temporary history, terminates with valid irreducible queries. ∎

We now obtain completeness for the general case, using the following two lemmas.

Lemma 6.13
If $\models^*_{\mathbf{Ł}} \Gamma_1|\Sigma \vdash^?_c \Delta_1; (\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ then $\models^*_{\mathbf{Ł}} \Gamma_1, m\Sigma|\emptyset \vdash^?_c \Delta_1; \{(\Gamma_2, n\Sigma \vdash^? \Delta_2), \ldots, (\Gamma_n, m\Sigma \vdash^? \Delta_n)\}$ for some $m \in \mathbb{N}$.

Proof. Follows from the fact that $\models^*_{\mathbf{Ł}} \Gamma_1|\Sigma \vdash^?_c \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ can be expressed as a logical consequence in Abelian logic, see [19] for details, which has the deduction theorem [20]. ∎

Lemma 6.14
If $\Gamma_1, \Sigma_1|\emptyset \vdash^?_c \Delta_1; \{(\Gamma_2, \Sigma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Sigma_n \vdash^? \Delta_n)\}$ succeeds in **FLPŁ** then $\Gamma_1|\Sigma \vdash^?_c \Delta_1; \{(\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$ succeeds in **FLPŁ** where $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$, and $\cup$ is set union.

Proof. A simple induction on the length of a derivation of $\Gamma_1, \Sigma_1|\emptyset \vdash^?_c \Delta_1; \{(\Gamma_2, \Sigma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n, \Sigma_n \vdash^? \Delta_n)\}$. ∎

Theorem 6.15 (Completeness of **FLPŁ**)
For $Q = \Gamma|\Sigma \vdash^?_c \Delta; H$, if $\models^*_{\mathbf{Ł}} Q$ then Q succeeds in **FLPŁ**.

Proof. Let $Q = \Gamma_1|\Sigma \vdash^?_c \Delta_1; (\Gamma_2 \vdash^? \Delta_2), \ldots, (\Gamma_n \vdash^? \Delta_n)\}$. If $\models^*_{\mathbf{Ł}} Q$ then, by Lemma 6.13, $\models^*_{\mathbf{Ł}} Q'$ where $Q' = \Gamma_1, m\Sigma|\emptyset \vdash^?_c \Delta_1; \{(\Gamma_2, n\Sigma \vdash^? \Delta_2), \ldots, (\Gamma_n, m\Sigma \vdash^? \Delta_n)\}$ for some $m \in \mathbb{N}$. Hence by Lemma 6.12, Q' succeeds in **FLPŁ**, but then by Lemma 6.14, so also does Q as required. ∎

Note finally that all the refinements made in Section 5, can also be made here. In particular, we can define more efficient rules introducing new propositional variables, as follows:

(or) From $\Gamma|\Sigma \vdash^?_c A \vee B, \Delta; H$ step to:
$\Gamma, y|\Sigma \vdash^?_c x, \Delta; H \cup \{(x \vdash^? y, A), (x \vdash^? y, B)\}$ where x and y are new

(red_l) From $\Gamma|\Sigma \vdash^?_c q, \Delta; H$ where $\Pi \to (A \vee q') \in \Gamma \cup \Sigma$ and $q' \in \{q\} \cup \overline{C}$ step to
$\Gamma', \Pi \to A|\Sigma \vdash^?_{c_1} q, \Delta; H$ and $q'|\Sigma \vdash^?_{c_2} \Pi, 2x; H \cup \{(\Gamma', 2x \vdash^? q, \Delta)\}$
where $\Gamma' = \Gamma - \{\Pi \to (A \vee q')\}$ if $\Pi \to (A \vee q') \in \Gamma$, otherwise $\Gamma' = \Gamma$,
x is new, and $c = min(c_1, c_2)$

(red_r) From $\Gamma|\Sigma \vdash^?_c q, \Delta; H \cup \{(\Gamma', \Pi \to (A \vee q) \vdash^? \Delta')\}$ step to:
$\Gamma \vdash^?_{c_1} q, \Delta; H \cup \{(\Gamma', \Pi \to A \vdash^? \Delta')\}$ and
$q \vdash^?_{c_2} \Pi, 2x; H \cup \{(\Gamma', 2x \vdash^? \Delta'), (\Gamma \vdash^? q, \Delta)\}$
where x is new, and $c = min(c_1, c_2)$

6.3 Examples

Below we discuss some examples of how **FLPŁ** can be used to design a logic programming language. In a realistic logic programming application, the clauses contained in the database would be first order statements. However, as noted above, we assume here that a restricted function-free language is used, and this means that for a finite domain we can translate such clauses into propositional logic by taking all possible instances as the permanent part of the database. Thus we assume we have a collection of atoms indexed by individual constants (written as first-order atomic formulae).

Example 6.16
Consider the following predicates:

- factory(x): x is a factory.
- produces(x, y): factory x produces substance y.
- da_fact(x): x is a dangerous factory.
- toxic(x): x is a toxic substance.
- close_to(x, y): x and y are close to each other.
- unhealthy(x): x is an unhealthy area.
- lives(x, y): x lives in y.
- aff_he(x): the health of x is affected.

factory(x), produces(x, y), and lives(x, y) are intended as crisp predicates, and the others as fuzzy. We define a permanent database Σ containing all instances over the set of individuals $\{a_1, a_2, p_1, p_2, v_1, v_2, john, mary\}$ of the following clauses:

$$\begin{array}{ll}
(K1_{x,y}) & (\text{factory}(x) \wedge \text{produces}(x, y) \wedge \text{toxic}(y)) \rightarrow \text{da\_fact}(x) \\
(K2_{x,y}) & (\text{da\_fact}(y) \odot \text{close\_to}(x, y)) \rightarrow \text{unhealthy}(x) \\
(K3_{x,y}) & (\text{lives}(x, y) \wedge \text{unhealthy}(y)) \rightarrow \text{aff\_he}(x)
\end{array}$$

together with the following collection of fuzzy facts, writing $a : A$ for $\underline{a} \rightarrow A$:

- $1 :$ factory(a_1).
- $1 :$ factory(a_2).
- $1 :$ produces(a_1, p_1).
- $1 :$ produces(a_2, p_2).
- $1 :$ lives$(john, v_1)$.
- $1 :$ lives$(mary, v_2)$.
- $.8 :$ toxic(p_1).
- $.3 :$ toxic(p_2).
- $.3 :$ close_to(a_1, v_1).
- $.7 :$ close_to(a_1, v_2).
- $.5 :$ close_to(a_2, v_1).
- $.8 :$ close_to(a_2, v_2).

The query "how dangerous is factory a_1?" can then be written as:

$$X \mid \Sigma \vdash_c^? \text{da\_fact}(a_1); \emptyset$$

The computation in **FLPŁ** steps by (red_l) to

$$X, \text{da\_fact}(a_1) \mid \Sigma \vdash^?_c \text{da\_fact}(a_1), \text{factory}(a_1) \wedge \text{produces}(a_1, p_1) \wedge \text{toxic}(p_1); H$$

where $H = \{(X \vdash^? \text{da\_fact}(a_1))\}$, then by (and) twice to

$$\begin{array}{l} X, \text{da\_fact}(a_1) \mid \Sigma \vdash^?_{c_1} \text{da\_fact}(a_1), \text{factory}(a_1); \{(X \vdash^? \text{da\_fact}(a_1))\} \\ X, \text{da\_fact}(a_1) \mid \Sigma \vdash^?_{c_2} \text{da\_fact}(a_1), \text{produces}(a_1, p_1); \{(X \vdash^? \text{da\_fact}(a_1))\} \\ X, \text{da\_fact}(a_1) \mid \Sigma \vdash^?_{c_3} \text{da\_fact}(a_1), \text{toxic}(p_1); \{(X \vdash^? \text{da\_fact}(a_1))\} \end{array}$$

where $c = min(c_1, c_2, c_3)$. The above queries succeed with $c_1 = c_2 = 1$ and $c_3 = 0.8$. E.g., for the latter:

$$X, \text{da\_fact}(a_1), \text{toxic}(p_1) \mid \Sigma \vdash^?_{c_3} \text{da\_fact}(a_1), \text{toxic}(p_1), 0.8; H'$$

where $H' = \{(X \vdash^? \text{da\_fact}(a_1)), (X, \text{da\_fact}(a_1) \vdash^? \text{da\_fact}(a_1), \text{toxic}(p_1))\}$. From this we get $c_3 = \max\{w \mid w - 1 \leq 0.8 - 1\} = 0.8$. Hence the original query succeeds with $c = min(1, 1, 0.8) = 0.8$.

Now consider the query "How much is Mary's health affected?", written as:

$$X \mid \Sigma \vdash^?_c \text{aff\_he}(mary); \emptyset$$

First we step by (red_l) to:

$$X, \text{aff\_he}(mary) \mid \Sigma \vdash^?_c \text{aff\_he}(mary), \text{lives}(mary, v_2) \wedge \text{unhealthy}(v_2); H$$

where $H = \{(X \vdash^? \text{aff\_he}(mary))\}$, then by (and) to

$$\begin{array}{l} X \mid \Sigma \vdash^?_{c_1} \text{lives}(mary, v_2); H \\ X \mid \Sigma \vdash^?_{c_2} \text{unhealthy}(v_2); H \end{array}$$

where $c = min(c_1, c_2)$. The former succeeds with $c_1 = 1$. To reduce the latter there are two choices corresponding to the two relevant instances of $(K2)$, to get the maximal value we must try both and then compare the two solutions. If we use $\text{da\_fact}(a_1) \odot \text{close\_to}(a_1, v_2) \rightarrow \text{unhealthy}(v_2)$, we get $c = 0.5$, since $0.5 = \max\{w \mid w - 1 \leq (0.8 - 1) + (0.7 - 1)\}$. If we use $\text{da\_fact}(a_2) \wedge \text{close\_to}(a_2, v_2) \rightarrow \text{unhealthy}(v_2)$, we get $c = 0.1 = \max\{w \mid w - 1 \leq (0.3 - 1) + (0.8 - 1)\}$. Thus the original query succeeds with $c = 0.5$.

Example 6.17
We can also express rules having numeric constraints in the body (similarly to *annotated logic programming* [12]), such as:

$$(K4_{x,y}) : (\text{lives}(x, y) \wedge (0.5 \rightarrow \text{unhealthy}(y)) \rightarrow \text{reg\_health\_checks}(x).$$

$0.5 \rightarrow \text{unhealthy}(x)$ can be read as "x is rather unhealthy". Thus the rule can be read as "if someone lives in a rather unhealthy area (that is the value of unhealthy is ≥ 0.5) then (s)he needs regular health checks". For example, if we let Σ' be the database of the previous example Σ expanded with all instances of $K4_{x,y}$, we have that $\Sigma' \vdash^? \text{reg\_health\_checks}(mary)$ succeeds, whereas $\Sigma' \vdash^? \text{reg\_health\_checks}(john)$ fails.

Example 6.18
We finally consider an example requiring disjunctive reasoning. Let the (permanent) database contain the following rules and facts: "if someone has good relational skills then (s)he is a good candidate for the marketing department or for the consulting department", "a young person cannot be a good candidate for the marketing department", "john is rather young and has good relational skills"; written as:

- $\mathrm{good\_rel}(x) \to (\mathrm{good\_Man}(x) \vee \mathrm{good\_Cons}(x))$
- $(\mathrm{young}(x) \wedge \mathrm{good\_Man}(x)) \to \underline{0}$
- $0.8 \to \mathrm{young}(john)$
- $0.9 \to \mathrm{good\_rel}(john)$

Let Σ be the set of the above clauses instantiated with john. We want to compute the maximal value for the goal $\mathrm{good\_Cons}(john)$, thus the query will be:

$$X \mid \Sigma \vdash^{?}_{c} \mathrm{good\_Cons}(john); \emptyset$$

The query succeeds with maximal value $c = 0.8$. However note that if $(\mathrm{young}(x) \wedge \mathrm{good\_Man}(x)) \to \underline{0}$ is replaced by

$$(\mathrm{young}(x) \odot \mathrm{good\_Man}(x)) \to \underline{0},$$

i.e. we use the t-norm $\odot$ to combine potentially conflicting information, we get a maximal value $c = 0.7$.

7 Concluding Remarks

In this paper we have defined a basic goal-directed calculus for Łukasiewicz logic **Ł** with a purely logical intepretation. We have subsequently refined this calculus in two directions. On the one hand, we have obtained more efficient and terminating calculi suitable for theorem proving. On the other, we have investigated extensions of the calculus suitable for defining fuzzy logic programming languages based on Rational Pavelka logic **RPL**. From the theorem proving perspective, the calculi presented here are a significant improvement on other automated reasoning methods for **Ł** proposed in the literature. Unlike the tableaux calculi of Hähnle [8] and Olivetti [25], and the resolution calculi of Wagner [32], and Mundici and Olivetti [24], each step in our calculi has an intuitive logical interpretation. Moreover, the goal-directed methodology both gives an algorithmic reading of the logic, and ensures that, rather than decomposing all formulae (as in the cited approaches), only formulae relevant to the current proof are treated. As for the mentioned tableaux calculi we can obtain a co-NP decision procedure for **Ł** using linear programming. We also provide purely logical terminating calculi. Note, however, that the complexity of these calculi is not optimal, as the length of derivation branches of irreducible hypersequents is not bounded polynomially. We intend to investigate methods for obtaining such a polynomial bound in future work, e.g. by refining the rules of the calculus and/or adopting a suitable control strategy.

In order to obtain calculi suitable for logic programming applications, we have had to extend our calculi in several directions. The basic goal-directed calculus is defined only for **Ł** with implication and falsity. Although other standard connectives are definable in this language, it is too limited for specifying fuzzy logic programs and deductive databases. We have therefore extended the calculi to handle formulae in a simple normal form allowing conjunctions and disjunctions, making a distinction, as is customary, between database and goal formulae. We have also introduced numerical values in the form of rational truth-constants into the object language, thereby obtaining Rational Pavelka logic **RPL**. This differs from standard approaches to fuzzy logic programming, where numerical data is introduced as extra-logical annotations on formulae. Another extension of the calculus is to deal with the *logical consequences* of a set of data, since this is the semantic notion of interest in deductive database and logic programming applications. Moreover, in fuzzy logic programming, one is often interested in a *graded* notion of logical consequence, i.e. we aim to compute to what *degree* a formula is entailed by a set of data. We have therefore modified our proof procedure in order to compute the maximal truth-value for which a goal is a logical consequence of a database. This further extension generalizes some Prolog-like systems proposed in the literature [13, 31, 30], (see also the "quantitative" variant of Prolog introduced by Van Emden, and developed by Kifer and Subrahmanian in [12]), where the database is essentially a set of (annotated) Horn clauses.

Finally, note that in this paper we have considered essentially only propositional Łukasiewicz logic. In future work we aim to investigate how our proof-methods might be extended to the first-order level. This extension is non trivial, as first-order **Ł** is not recursively axiomatizable. Nontheless, it is worth investigating fragments for which there exist complete proof-systems, and which might be suitable for logic programming applications.

Acknowledgements: We would like to thank the anonymous referee for useful and constructive criticisms that have helped to improve the paper.

References

[1] S. Aguzzoli and A. Ciabattoni. Finiteness in infinite-valued logic. *Journal of Logic, Language and Information*, 9(1):5–29, 2000.

[2] R. Cignoli, I. M. L. D'Ottaviano, and D. Mundici. *Algebraic Foundations of Many-Valued Reasoning*, volume 7 of *Trends in Logic*. Kluwer, Dordrecht, November 1999.

[3] C. V. Damásio and P. Pereira. Hybrid probabilistic logic programs as residuated logic programs. *LNCS*, 1919:57–72, 2000.

[4] D. Dubois, J. Lang, and H. Prade. Possibilistic logic. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3: Nonmonotonic Reasoning and Uncertain Reasoning*, pages 439–513. Oxford University Press, Oxford, 1994.

[5] D. Gabbay and N. Olivetti. *Goal-directed Proof Theory*. Kluwer Academic Publishers, 2000.

[6] D. Gabbay and N. Olivetti. Goal oriented deductions. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 9, pages 199–285. Kluwer Academic Publishers, second edition, 2002.

[7] D. Gabbay and Uwe Reyle. N-Prolog: An extension of Prolog with hypothetical implication. *Journal of Logic Programming*, 4:319–355, 1984.

[8] R. Hähnle. *Automated Deduction in Multiple-Valued Logics*. Oxford University Press, 1993.

[9] P. Hájek. *Metamathematics of Fuzzy Logic*. Kluwer, Dordrecht, 1998.

[10] J. Harland and D. Pym. The uniform proof-theoretic foundation of linear logic programming. In *Proc. of the 1991 International Logic Programming Symposium*, pages 304–318, 1991.

[11] J. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110:327–365, 1994.

[12] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. Log. Program.*, 12(3&4):335–367, 1992.

[13] F. Klawonn and R. Kruse. A Łukasiewicz logic based Prolog. *Mathware & Soft Computing*, 1(1):5–29, 1994.

[14] J. Łukasiewicz. *Jan Łukasiewicz, Selected Writings*. North-Holland, 1970. Edited by L. Borowski.

[15] J. Łukasiewicz and A. Tarski. Untersuchungen über den Aussagenkalkül. *Comptes Rendus des Séances de la Societé des Sciences et des Lettres de Varsovie, Classe III*, 23, 1930. Reprinted and translated in [14].

[16] R. McNaughton. A theorem about infinite-valued sentential logic. *Journal of Symbolic Logic*, 16(1):1–13, 1951.

[17] G. Metcalfe, N. Olivetti, and D. Gabbay. Goal-directed calculi for Gödel-Dummett logics. In M. Baaz and J. A. Makowsky, editors, *Proceedings of CSL 2003*, volume 2803 of *LNCS*, pages 413–426. Springer, 2003.

[18] G. Metcalfe, N. Olivetti, and D. Gabbay. Goal-directed methods for Łukasiewicz logics. In J. Marcinkowski and A. Tarlecki, editors, *Proceedings of CSL 2004*, volume 3210 of *LNCS*, pages 85–99. Springer, 2004.

[19] G. Metcalfe, N. Olivetti, and D. Gabbay. Sequent and hypersequent calculi for abelian and Łukasiewicz logics. To appear in ACM Transactions on Computational Logic, 2005.

[20] R. K. Meyer and J. K. Slaney. Abelian logic from A to Z. In G. Priest et al., editor, *Paraconsistent Logic: Essays on the Inconsistent*, pages 245–288. Philosophia Verlag, 1989.

[21] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[22] D. Mundici. Satisfiability in many-valued sentential logic is NP-complete. *Theoretical Computer Science*, 52(1-2):145–153, 1987.

[23] D. Mundici. The logic of Ulam's game with lies. In C. Bicchieri and M.L. Dalla Chiara, editors, *Knowledge, belief and strategic interaction*, pages 275–284. Cambridge University Press, 1992.

[24] D. Mundici and N. Olivetti. Resolution and model building in the infinite-valued calculus of Łukasiewicz. *Theoretical Computer Science*, 200(1–2):335–366, 1998.

[25] N. Olivetti. Tableaux for Łukasiewicz infinite-valued logic. *Studia Logica*, 73(1), 2003.

[26] J. Pavelka. On fuzzy logic I,II,III. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 25:45–52,119–134,447–464, 1979.

[27] D. J. Pym and J. A. Harland. The uniform proof-theoretic foundation of linear logic programming. *Journal of Logic and Computation*, 4(2):175–206, 1994.

[28] A. Rose and J. B. Rosser. Fragments of many-valued statement calculi. *Transactions of the American Mathematical Society*, 87:1–53, 1958.

[29] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1987.

[30] D. Smutná-Hlinená and P. Vojtás. Graded many-valued resolution with aggregation. *Fuzzy Sets and Systems*, 143(1):157–168, 2004.

[31] P. Vojtás. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361–370, 2001.

[32] H. Wagner. A new resolution calculus for the infinite-valued propositional logic of Łukasiewicz. In Ricardo Caferra and Gernot Salzer, editors, *Int. Workshop on First-Order Theorem Proving (FTP'98)*, Technical Report E1852-GS-981, pages 234–243. Technische Universität Wien, Austria, 1998.

Received 31 December 2004