Research Report

Artificial Intelligence Research Institute (IIIA)
Spanish National Research Council (CSIC)

Barcelona - Nov. 2013

NiBLoS: a general SMT-based solver for logics of BL-chains

Amanda Vidal Wandelmer

Join work with

Félix Bou and Lluìs Godo

Contents

1. Introduction

To the best of our knowledge, little attention has been paid to the development of a solver for systems of mathematical fuzzy logic. Some theoretical works about computations (mainly for Łukasiewicz logic) have been presented and there is also an important number of studies on complexity and proof theory on fuzzy logics too, but a working application that copes with the most common fuzzy logics had still not been implemented.

We consider this is a problem that limits the use of fuzzy logics in real applications, mainly in the field of Artificial Intelligence, where several theoretical proposes using these logics have long been presented. On the other hand, no solvers for infinitely many valued fuzzy logics have been implemented in any way, which is very interesting from a theoretical point of view for the logician community as a practical challenge for testing the theoretical results, finding concrete examples and working with infinitely valued (continuous) concepts.

In [ABMV12], a new approach for implementing a theorem prover for Łukasiewicz, Gödel and Product fuzzy logics using Satisfiability Modulo Theories was proposed. The idea of using Satisfiability Modulo Theories solvers for this problem was new and the results were interesting for Łukasiewicz and Gödel because the results were optimistically efficient. Also, the modularity this approach enjoys allows to naturally cope with several fuzzy logics. However, in the case of Product logic, the results were far from being satisfactory.

In the present work, we give details on the theoretical basis necessary for the extension of Ansótegui's work to an application that is able to solve problems over a wider family of BL-chains. We present the implementation details and the obtained results of a new solver which is a generalization of the one shown in [ABMV12], that allows the use of a wide family of logics and the resolution of a bigger set of problems, and that works (on the problems for which any comparative is possible) more efficiently that the previous existent reasoners. Our research has led to the implementation of a reasoner that is able to more efficiently cope with all continuous t-norm based fuzzy logics and with a set of logics defined by certain BL-chains (the ones that are expressed as a sum of the three main Ł, G and Π t-norms and finitely-valued logics of the families G_n and $Ł_n$) and also with Basic Fuzzy Logic BL. Its tasks have also been generalized and the reasoner can perform satisfiability, theoremhood and logical consequence checks for any of a wide family of these fuzzy logics. An intermediate version of this research was accepted in the Scalable Uncertainty Management

(SUM) 2012 Conference, and has been published in its proceedings in September 2012 [VBG12].

This report is organized as follows:

In Chapter 2 a description of Basic Fuzzy Logics is given, and the theoretical basis necessary for the design of the final solver are detailed.

In Chapter 3 an overview on the complexity of the worst case scenario for the previously presented fuzzy logics is given so the interested reader can have in mind the kind of problems we are working with.

In Chapter 4 it is shown a wide study about Satisfiabilty Modulo Theories, comprehending the several existing theoretical approaches implementation techniques and the reasons that lead to the use of SMT and specifically, of the SMT-solver we have finally chosen, for the design and implementation of the fuzzy logics solver.

In Chapter 5, a detailed description of the main result of this research is presented, namely a complete specification of the continuous t-norms based fuzzy logics solver we have developed. Details on the theoretical approach, the technical implementation and the obtained results are shown. Also we comment here the results about an alternative solver that works over discrete algebras.

Finally, in Chapter 6 a review on existing related works is given, and also a section presenting the conclusions reached after this research and the future works continuing this line we propose.

1.1. **Acknowledgements**

The authors acknowledge support of the Spanish projects EdeTRI (TIN2012-39348-C02-01) and AT (CONSOLIDER CSD 2007- 0022). Amanda Vidal is supported by a CSIC grant JAE Predoc.

2. Basic Logic and its extensions

Hájek presented in [Háj98] a basic logic (BL) that was a common fragment of the most well known existing fuzzy logics until that date, namely Łukasiewicz, Gödel and Product logic. It is well known that the variety $\mathcal{BL}$ of BL-algebras is the equivalent algebraic semantics for Hájek's basic fuzzy logic BL. Thus, subvarieties of $\mathcal{BL}$ naturally correspond to schematic extensions of BL, i.e., to sets that are closed under substitution and Modus Ponens. The algebraic counterparts of the three logics commented above are the varieties of MV-algebras, Gödel-algebras and Product-algebras (respectively denoted by $\mathcal{MV}$, $\mathcal{G}$ and Π).

A second reason for introducing BL was the search of the logic of all continuous t-norms and their residuals. In [CEGT00] it is shown that BL is in fact complete with respect to the class of the so called t-norm BL-algebras.

In this chapter, we will give an overview of the most important definitions for understanding and working with BL and its main schematic extension logics and its algebraic counterparts, i.e. BL-algebras and its main subvarieties. Also, we will provide several important results about these logics that will be later applied.

2.1. BL-algebras

The logic BL was initially presented as an axiomatic system later proved complete with respect to a class of algebras.

The class of algebras to which BL is complete is defined as follows:

Definition 2.1. A **BL-algebra** is an algebra

$$\mathbf{A} = \langle A, *, \to, \vee, \wedge, \bot, \top \rangle$$

where

(1) $\langle A, \vee, \wedge, \bot, \top \rangle$ is a lattice with largest element $\top$ and least element $\bot$ (with respect to the lattice order $\leq$),

(2) $\langle A, *, \top \rangle$ is a commutative semigroup with unit element $\top$, i.e. for all $a, b, c \in A$ it holds that
   - $a * b = b * a$
   - $(a * b) * c = a * (b * c)$
   - $\top * a = a$,

(3) $*$ and $\to$ form a residuated pair, i.e.

$$\text{for all } a, b, c, \quad a * b \leq c \quad \text{iff} \quad b \leq (a \to c),$$

(4) for all $a, b \in A$, $\quad a \wedge b = a * (a \to b)$,

(5) for all $a, b \in A$, $\quad (a \to b) \vee (b \to a) = \top$.

Remark 2.2. Extra operations are defined from the existing ones as follows:

$$\begin{aligned} \neg x &:= x \to \bot \\ x \leftrightarrow y &:= (x \to y) \wedge (y \to x) \end{aligned}$$

A more general concept, namely that of hoops, concerning BL-algebras will be used in followings sections. The natural idea is to present an algebra with operations $*, \to$ that maintains some of the characteristics of the BL-algebras, but might not have a minimum. It has been proved to be useful in the study of BL-algebras.

Definition 2.3.

(1) A **hoop** is an algebra $\mathbf{A} = \langle A, \cdot, \to, \top \rangle$ such that $\langle A, \cdot, \top \rangle$ is a commutative monoid and for all $a, b, c \in A$
- $a \to a = \top$
- $a \cdot (a \to b) = b \cdot (b \to a)$
- $a \to (b \to c) = (a \cdot b) \to c$

(2) A **bounded hoop** is an algebra $\mathbf{A} = \langle A, \cdot, \to, \bot, \top \rangle$ such that $\langle A, \cdot, \to, \top \rangle$ is a hoop and $\bot \leq a$ for all $a \in A$.

(3) A **basic hoop** is a hoop that satisfies the equation

$$\big(((x \to y) \to z) \cdot ((y \to x) \to z)\big) \to z = \top$$

This also leads to an alternative characterization of BL-algebras as follows:

Lemma 2.4. *Let* $\mathbf{A} = \langle A, *, \to, \vee, \wedge, \bot, \top \rangle$ *be an algebra. Then* $\mathbf{A}$ *is a* BL*-algebra iff* $\langle A, *, \to, \bot, \top \rangle$ *is a bounded basic hoop and*

$$\begin{aligned} x \wedge y &= x * (x \to y) \\ x \vee y &= ((x \to y) \to y) \wedge ((y \to x) \to x) \end{aligned}$$

We will denote the variety of BL-algebras by $\mathcal{BL}$. There are three important subvarieties of $\mathcal{BL}$.

Definition 2.5.

(1) An **MV-algebra** (Many-Valued-algebra) is a BL-algebra in which the identity

$$\neg\neg x = x$$

is valid. We will denote by $\mathcal{MV}$ the variety of MV-algebras.

(2) A **G-algebra** (Gödel-algebra) is a BL-algebra in which the identity

$$x * y = x \wedge y$$

is valid. We will denote by $\mathcal{G}$ the variety of G-algebras.

(3) A **Π-algebra** (Product-algebra) is a BL-algebra in which the identities

$$\begin{aligned} x \wedge \neg x &= \bot \\ \neg\neg z \to ((x * z \to y * z) \to (x \to y)) &= \top \end{aligned}$$

are valid. We will denote by $\mathcal{P}$ the variety of Π-algebras.

2.2. **BL-chains**

We are interested in studying a concrete type of BL-algebras called BL-chains. This name denotes the classical meaning of chain of totally ordered set, having that a BL-chain is a BL-algebra whose universe is totally ordered by the order defined through $\vee, \wedge$ ($a \leq b$ iff $a \wedge b = a$). We are interested in the class of BL-chains because BL is complete with respect to it. The completeness of BL with respect to the class of BL-chains comes from the well known fact ([Háj98, Lem.2.3.16]) that each BL-algebra is a subalgebra of the direct product of a system of BL-chains.

For any BL-chain whose universe is the real unit interval, $\mathbf{A} = \langle [0,1], *, \to, \vee, \wedge, \bot, \top \rangle$, the operation $*$ is called a continous t-norm. We can give an explicit definition considering the conditions from Definition 2.1 that concern $*, \to$ in this universe, i.e.

Definition 2.6. A **continuous t-norm** $*$ is a continuous mapping from $[0,1]^2$ into $[0,1]$ (in the classical topological sense) such that

(1) $*$ is commutative and associative, i.e, $\forall a, b, c \in [0,1]$,
   - $a * b = b * a$,
   - $(a * b) * c = a * (b * c)$.

(2) $*$ is non-decreasing in both components, i.e., $\forall a, b, c \in [0,1]$,
   - $a \leq b$ implies $a * c \leq b * c$,
   - $a \leq b$ implies $c * a \leq c * b$.

(3) $1 * a = a$ and $0 * a = 0$ $\forall a \in [0,1]$

For each $*$ operation of a BL-algebra a unique *residuum* is implicitly defined (from point (3) in Definition 2.1). An explicit specification is given by:

Lemma 2.7. *Let* $\mathbf{A} = \langle A, *, \to, \vee, \wedge, \bot, \top \rangle$ *be a* BL*-algebra. Then, the* $\to$ *operation, called the residuum of* $*$*, is uniquely determined as the function* $\to\colon A \times A \longrightarrow A$ *such that for all* $a, b \in A$*,*

$$a \to b = \sup\{c : a * c \leq b\}$$

The BL-chains defined over $[0,1]$ are called **standard** BL**-chains**, and the class of all standard BL-chains will be denoted by $\mathcal{BL}st$.

Definition 2.8. Let $*$ be a continuous t-norm and $\to$ its residuum. Then, the algebra $[0,1]_* := \langle [0,1], *, \to, \vee, \wedge, \bot, \top \rangle$ (where $\vee = max$, $\wedge = min$) is a standard BL-chain, called the standard ***-algebra**.

The class of standard BL-chains is a very important family of BL-algebras because it is proved (see [CEGT00]) that BL is complete with respect to the standard BL-chains. Also it is proven that each one of the subvarieties of BL-algebras defined above ($\mathcal{MV}, \mathcal{G}$ and $\mathcal{P}$) is the variety generated by some standard BL-chain that will be detailed below.

Here some interesting examples of BL-chains are introduced.

2.2.1. *Concrete Examples of* BL*-chains.*

Example 2.9. Łukasiewicz BL-chains

(1) We denote by $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ the MV-algebra $\langle [0,1], *_{\text{Ł}}, \to_{\text{Ł}}, \vee, \wedge, \bot, \top \rangle$ where

$$\begin{aligned} x *_{\text{Ł}} y &:= max\{0, x+y-1\} \\ x \to_{\text{Ł}} y &:= \begin{cases} 1 & \text{if } x \leq y, \\ 1-x+y & \text{otherwise} \end{cases} \\ x \vee y &:= max\{x,y\} \\ x \wedge y &:= min\{x,y\} \\ \bot &:= 0 \\ \top &:= 1 \end{aligned}$$

In the literature, $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ is called the standard MV-algebra and $*_{\text{Ł}}$, $\to_{\text{Ł}}$ are called the Łukasiewicz t-norm and its residuum. It holds that $\mathcal{MV} = V([\mathbf{0},\mathbf{1}]_{\mathbf{Ł}})$.

(2) Let us consider the subalgebra of $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ whose domain is the set $\{0, \frac{1}{n-1}, \frac{2}{n-1} ..., 1\}$. We define

$$\mathbf{Ł_n} := \langle \{0, 1, ..., n-1\}, *_{\text{Ł}_n}, \to_{\text{Ł}_n}, \vee, \wedge, \bot, \top \rangle$$

to be the BL-algebra isomorphic to the previous one through the isomorphism $i_n : \{0, 1, ..., n-1\} \longrightarrow \{0, \frac{1}{n-1}, \frac{2}{n-1} ..., 1\}$, $i_n(x) = x \cdot (n-1)$, that is $\mathbf{Ł_n}$ is the BL-algebra with finite

universe $\{0, 1, ..., n-1\}$ where

$$\begin{aligned} x *_{\mathrm{L}_n} y &\coloneqq max\{0, x+y-(n-1)\} \\ x \to_{\mathrm{L}_n} y &\coloneqq \begin{cases} n-1 & \text{if } x \leq y, \\ n-1-x+y & \text{otherwise} \end{cases} \\ x \vee y &\coloneqq max\{x, y\} \\ x \wedge y &\coloneqq min\{x, y\} \\ \bot &\coloneqq 0 \\ \top &\coloneqq n-1 \end{aligned}$$

In particular, $\mathbf{Ł_2}$ coincides with the two valued Boolean algebra, which is also denoted by **2**.

Example 2.10. Gödel BL**-chains**

(1) We will denote by $[\mathbf{0}, \mathbf{1}]_{\mathbf{G}}$ the G-algebra $\langle [0,1], *_G, \to_G, \vee, \wedge, \bot, \top \rangle$ where

$$\begin{aligned} x *_G y &\coloneqq min\{x, y\} \\ x \to_G y &\coloneqq \begin{cases} 1 & \text{if } x \leq y, \\ y & \text{otherwise} \end{cases} \\ x \vee y &\coloneqq max\{x, y\} \\ x \wedge y &\coloneqq min\{x, y\} \\ \bot &\coloneqq 0 \\ \top &\coloneqq 1 \end{aligned}$$

In the literature, $[\mathbf{0}, \mathbf{1}]_{\mathbf{G}}$ is also called the standard G-algebra and $*_G$, $\to_G$ are called the Gödel t-norm and its residuum. It holds that $\mathcal{G} = V([\mathbf{0}, \mathbf{1}]_{\mathbf{G}})$.

(2) Let us consider the subalgebra of $[\mathbf{0}, \mathbf{1}]_{\mathbf{G}}$ whose domain is the set $\{0, \frac{1}{n-1}, \frac{2}{n-1}..., 1\}$. We define

$$\mathbf{G_n} \coloneqq \langle \{0, 1, ..., n-1\}, *_{G_n}, \to_{G_n}, \vee, \wedge, \bot, \top \rangle$$

to be the BL-algebra isomorphic to the previous one through the isomorphism $i : \{0, 1, ..., n-1\} \longrightarrow \{0, \frac{1}{n-1}, \frac{2}{n-1}..., 1\}$, $i(x) = x \cdot (n-1)$, that is $\mathbf{G_n}$ is the BL-algebra with finite

universe $\{0, 1, ..., n-1\}$ where

$$\begin{aligned} x *_{G_n} y &:= min\{x, y\} \\ x \to_{G_n} y &:= \begin{cases} n-1 & \text{if } x \leq y, \\ y & \text{otherwise} \end{cases} \\ x \vee y &:= max\{x, y\} \\ x \wedge y &:= min\{x, y\} \\ \bot &:= 0 \\ \top &:= n-1 \end{aligned}$$

Example 2.11. Product BL**-chains and hoops**

(1) We will denote by $[\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$ the Π-algebra $\langle [0,1], *_\Pi, \to_\Pi, \vee, \wedge, \bot, \top \rangle$ where

$$\begin{aligned} x *_\Pi y &:= x \cdot y \\ x \to_\Pi y &:= \begin{cases} 1 & \text{if } x \leq y, \\ \frac{y}{x} & \text{otherwise} \end{cases} \\ x \vee y &:= max\{x, y\} \\ x \wedge y &:= min\{x, y\} \\ \bot &:= 0 \\ \top &:= 1 \end{aligned}$$

In the literature, $[0,1]_\Pi$ is also called the standard Π-algebra, and $*_\Pi$, $\to_\Pi$ are called the Product t-norm and its residuum (Goguen implication). It holds that $\mathcal{P} = V([\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}})$.

(2) Let us consider the subalgebra as a hoop of $[\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$ whose domain is $(0, 1]$. We define $(\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}} = \langle (0,1], *_\Pi, \to_\Pi, \vee, \wedge, \top \rangle$ to be that subalgebra. Notice it is an unbounded basic hoop.

(3) We denote by $\mathbf{Z}^-_\bullet$ the Π-algebra $\langle \mathbb{Z}^- \cup \{-\infty\}, *_{\mathbf{Z}^-_\bullet}, \to_{\mathbf{Z}^-_\bullet}, \vee, \wedge, \bot, \top \rangle$ where

$$
\begin{aligned}
\mathbb{Z}^- &:= \{z \in \mathbb{Z} : z \leq 0\} \\
x *_{\mathbf{Z}^-_\bullet} y &:= \begin{cases} -\infty & \text{if either } x = -\infty \text{ or } y = -\infty, \\ x + y & \text{otherwise.} \end{cases} \\
x \to_{\mathbf{Z}^-_\bullet} y &:= \begin{cases} 0 & \text{if } x \leq y, \\ -\infty & \text{if } y = -\infty \\ y - x & \text{otherwise} \end{cases} \\
x \vee y &:= \begin{cases} -\infty & \text{if } x = y = -\infty \\ max\{x, y\} & \text{otherwise} \end{cases} \\
x \wedge y &:= \begin{cases} -\infty & \text{if either } x = -\infty \text{ or } y = -\infty \\ min\{x, y\} & \text{otherwise} \end{cases} \\
\bot &:= -\infty \\
\top &:= 0
\end{aligned}
$$

It also holds that $\mathcal{P} = V(\mathbf{Z}^-_\bullet)$.

(4) Let us consider the subalgebra as a hoop of $\mathbf{Z}^-_\bullet$ whose domain is $\mathbb{Z}^-$. We define $\mathbf{Z}^- := \langle \mathbb{Z}^-, *_{\mathbf{Z}^-}, \to_{\mathbf{Z}^-}, \vee, \wedge, \top \rangle$ to be that subalgebra, i.e., the basic unbounded hoop with universe $\mathbb{Z}^-$ such that

$$
\begin{aligned}
x *_{\mathbf{Z}^-} y &:= x + y \\
x \to_{\mathbf{Z}^-} y &:= \begin{cases} 0 & \text{if } x \leq y, \\ y - x & \text{otherwise} \end{cases} \\
x \vee y &:= max\{x, y\} \\
x \wedge y &:= \min\{x, y\} \\
\top &:= 0
\end{aligned}
$$

2.2.2. *BL-chains constructions.* It is not only interesting to specify BL-chains through its explicit definition, but also to be able to build BL-chains from a family of other BL-chains or from an adequate family of hoops. For this, we define the notion of ordinal sum operation.

The original definition of ordinal sum was the one for continuous t-norms. The one we provide here adapts this one to a wider universe, non only standard but general BL-chains.

Definition 2.12. ([CT05]) Let $\langle I, \leq \rangle$ be a chain with a least element 0 and largest element 1. For each $i \in I$, let

$$i^+ := \begin{cases} \inf\{j \in I : i < j\} & \text{if it exists,} \\ i & \text{otherwise.} \end{cases}$$

Let $\{\mathbf{A_i}\}_{i \in I}$ be a family of BL-chains, where $\mathbf{A_i} = \langle A_i, *_i, \to_i, \vee_i, \wedge_i, \bot_i, \top_i \rangle$ and such that

- For all $i < j$, $A_i \cap A_j = \emptyset$ if $j \neq i^+$,
- For $i \neq 1$, $A_i \cap A_{i^+} = \{\top_i\} = \{\bot_{i^+}\}$.

Then we define its **ordinal sum** as the BL-chain

$$\boxplus_{i \in I} \mathbf{A_i} = \langle \bigcup_{i \in I} A_i, *, \to, \vee, \wedge, \bot, \top \rangle$$

where

$$x * y := \begin{cases} x *_i y & \text{if } x, y \in A_i \\ x & \text{if } x \in A_i,\ y \in A_j \text{ and } i <_I j \\ y & \text{otherwise} \end{cases}$$

$$x \to y := \begin{cases} \top & \text{if } x, y \in A_i \text{ and } x \leq_i y \text{ or } x \in A_i, y \in A_j \text{ and } i <_I j, \\ x \to_i y & \text{if } x, y \in A_i \text{ and } x >_i y \\ y & \text{otherwise} \end{cases}$$

$$x \vee y := \begin{cases} x \vee_i y & \text{if } x, y \in A_i \\ x & \text{if } x \in A_i,\ y \in A_j \text{ and } i >_I j \\ y & \text{otherwise} \end{cases}$$

$$x \wedge y := \begin{cases} x \wedge_i y & \text{if } x, y \in A_i \\ x & \text{if } x \in A_i,\ y \in A_j \text{ and } i <_I j \\ y & \text{otherwise} \end{cases}$$

$$\bot := \bot_0$$

$$\top := \top_1$$

Remark 2.13.

(1) If $I = \{0, 1, ..., n\}$ then we can write $\mathbf{A_0} \boxplus ... \boxplus \mathbf{A_n}$ instead of $\boxplus_{i \in I} \mathbf{A_i}$.

(2) With this definition we assume the BL-chains are disjoint. If we are considering ordinal sums of non-disjoint BL-chains, we will just assume the use of isomorphic copies of these algebras. For example, when saying $\boxplus_{i \in I} [\mathbf{0}, \mathbf{1}]_{\mathbf{\L}}$, we consider this to be the

sum of I copies of the $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ BL-chain, each one of them with a different universe but isomorphic to $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$.

An alternative notion of ordinal sum is the one that ranges over hoops, proposed by Büchi and Owens and presented in [Fer92]. This alternative ordinal sum will be used in the implementation of our solver.

The definition showed here is a variation over the usual definition of hoops ordinal sum, where the hoops must be pairwise disjoin (except from $\top$ element). Here, that condition is integrated in the definition working with pairs where the second element just refers to the component identification.

Definition 2.14. (cf. [AFM07]) Let I be an initial segment of $\mathbb{N}$. For each $i \in I$, let $\mathbf{A}_i = \langle A_i, *_i, \to_i, \top_i \rangle$ be a linearly ordered hoop and suppose $\mathbf{A_0}$ is also bounded (i.e. a BL-chain).

Then we define its **ordinal sum** as the BL-algebra

$$\bigoplus_{i\in I} \mathbf{A_i} = \langle K, *, \to, \vee, \wedge, \bot, \top \rangle$$

where

$$K \coloneqq \{\top^K\} \cup \bigcup_{i\in I} \{(A_i \setminus \{\top_i\}) \times \{i\}\}$$

with $\top^K$ being an arbitrary element not belonging to $\bigcup_{i\in I}\{(A_i\setminus\{\top_i\})\times\{i\}$ that will be the $\top$ element in K (for instance, assume $\top^K \coloneqq \langle 0, -1\rangle$). The order in K is defined by the reversed lexicographic order with last element $\top^K$, i.e.

$$\langle a, i_1\rangle \leq \langle b, i_2\rangle \text{ iff } (\langle b, i_2\rangle = \top^K) \text{ or } (i_1 < i_2) \text{ or } (i_1 = i_2 \text{ and } a \leq_{i_1} b)$$

and the operations are defined by

$$\langle x, i_1\rangle * \langle y, i_2\rangle \coloneqq \begin{cases} \langle x *_{i_1} y, i_1\rangle & \text{if } i_1 = i_2, \\ min\{\langle x, i_1\rangle, \langle x, i_2\rangle\} & \text{otherwise.} \end{cases}$$

$$\langle x, i_1\rangle \to \langle y, i_2\rangle \coloneqq \begin{cases} \top^K & \text{if } \langle x, i_1\rangle \leq \langle y, i_2\rangle, \\ \langle x \to_{i_1} y, i_1\rangle & \text{if } i_1 = i_2 \text{ and } y <_{i_1} x, \\ \langle y, c_2\rangle & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \langle x, i_1\rangle \vee \langle y, i_2\rangle &\coloneqq max\{\langle x, i_1\rangle, \langle y, i_2\rangle\} \\ \langle x, i_1\rangle \wedge \langle y, i_2\rangle &\coloneqq min\{\langle x, i_1\rangle, \langle y, i_2\rangle\} \\ \bot &\coloneqq \bot_0 \\ \top &\coloneqq \top^K \end{aligned}$$

Remark 2.15. If $I = \{0, 1, ..., n\}$ then we can write $\mathbf{A_0} \oplus ... \oplus \mathbf{A_n}$ instead of $\bigoplus_{i\in I} \mathbf{A_i}$.

2.3. **Logics**

The set of formulas on BL (from now on, Fm) is built over an alphabet with countably many propositional variables, basic connectives $0, \&, \Rightarrow$ and defined connectives $1, \neg, \wedge, \vee, \Leftrightarrow$ where:

$$\begin{array}{rcl} 1 & \text{is} & 0 \Rightarrow 0 \\ \neg\varphi & \text{is} & \varphi \Rightarrow 0 \\ \varphi \wedge \psi & \text{is} & \varphi\&(\varphi \Rightarrow \psi) \\ \varphi \vee \psi & \text{is} & ((\varphi \Rightarrow \psi) \Rightarrow \psi) \wedge ((\psi \Rightarrow \varphi) \Rightarrow \varphi) \\ \varphi \Leftrightarrow \psi & \text{is} & (\varphi \Rightarrow \psi)\&(\psi \Rightarrow \varphi) \end{array}$$

Given a BL-chain $\mathbf{L}$, an $\mathbf{L}$-evaluation of propositional variables is a mapping e that assigns to each propositional variable p an element of $\mathbf{L}$. Each evaluation of propositional variables extends uniquely to propositional formulas as follows:

$$\begin{array}{rcl} e(0) & \coloneqq & \bot \\ e(\varphi\&\psi) & \coloneqq & e(\varphi) * e(\psi) \\ e(\varphi \Rightarrow \psi) & \coloneqq & e(\varphi) \to e(\psi) \end{array}$$

Having the language, we can associate a logic (and its theorems) to a BL-chain.

Definition 2.16. Let $\mathbf{A}$ be a BL-chain.

(1) Given Γ a finite set of formulas (notation $\Gamma \subseteq_\omega Fm$) and a formula φ,

$$\Gamma \models_{\mathbf{A}} \varphi \quad \text{if} \quad \forall e \text{ } \mathbf{A}\text{-evaluation, if } e[\Gamma] \subseteq \{\top\} \text{ then } e(\varphi) = \top$$

(2) The **logic** of $\mathbf{A}$ is defined as

$$\Lambda(\mathbf{A}) \coloneqq \{\langle \Gamma, \varphi\rangle : \ \Gamma \models_{\mathbf{A}} \varphi, \ \varphi \in Fm, \ \Gamma \subseteq_\omega Fm\}$$

(3) We say φ is a **theorem** of $\mathbf{A}$ if $\emptyset \models_{\mathbf{A}} \varphi$. We define

$$Th(\mathbf{A}) \coloneqq \{\varphi : \ \varphi \text{ is a theorem of } \mathbf{A}\}$$

(4) We say an $\mathbf{A}$-evaluation e is a **model** of φ in $\mathbf{A}$ if in $\mathbf{A}$ it holds $e(\varphi) = \top$. If such model exists, φ is **satisfiable** in $\mathbf{A}$.

Given a finite set of formulas $\{\varphi_i\}_{i\in I}$, we say it is **satisfiable** in $\mathbf{A}$ if there exists an $\mathbf{A}$-evaluation e such that e is a model of φ_i in $\mathbf{A}$ for each $i \in I$.

(5) We define an **equation** on Fm as a triplet $\varphi \ R \ \psi$ where $\varphi, \psi \in Fm$ and $R \in \{=, >\}$.

We say an $\mathbf{A}$-evaluation e is a **model of an equation** $\varphi \ R \ \psi$

in $\mathbf{A}$ if in $\mathbf{A}$ it holds $e(\varphi) R^{\mathbf{A}} e(\psi)$. If such model exists, $\varphi\ R\ \psi$ is **satisfiable** in $\mathbf{A}$.

Given a finite set of equations in Fm $\{eq_i\}_{i\in I}$, we say it is **satisfiable** in $\mathbf{A}$ if there exists an $\mathbf{A}$-evaluation e such that e is a model of eq_i in $\mathbf{A}$ for each $i \in I$.

The most well known cases of logics associated to a BL-chain are the following:

Example 2.17.

(1) $\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{Ł}})$ is called **Łukasiewicz Logic**;
(2) $\Lambda(\mathbf{Ł_n})$ is called **n-valued Łukasiewicz Logic**;
(3) $\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{G}})$ is called **Gödel Logic**;
(4) $\Lambda(\mathbf{G_n})$ is called **n-valued Gödel Logic**;
(5) $\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}})$ is called **Product Logic**.

An interesting result related with this work (see Section 2.4) is the following.

Theorem 2.18. *([CT00])*

$$Th([\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}) = Th(\mathbf{Z}^-_{\bullet})$$

The extension of the concept of logic to a class of algebras is the natural one.

Definition 2.19. Given a class $\mathcal{K}$ of algebras, we define

$$\Lambda(\mathcal{K}) \coloneqq \bigcap_{\mathbf{A}\in\mathcal{K}} \Lambda(\mathbf{A})$$
$$Th(\mathcal{K}) \coloneqq \bigcap_{\mathbf{A}\in\mathcal{K}} Th(\mathbf{A})$$

And given $\Gamma \subseteq_\omega Fm,\ \varphi \in Fm$,

$$\Gamma \models_{\mathcal{K}} \varphi \text{ iff } \Gamma \models_{\mathbf{A}} \varphi \text{ for each } \mathbf{A} \in \mathcal{K}$$

The most well known logic associated to a class of algebras is, as we said before, Hájeks Basic Logic

Example 2.20. $\Lambda(\mathcal{K})$ where $\mathcal{K}$ is the class of all standard BL-chains is called the **Basic Logic** BL.

The following are two folklore results concerning logics and their theorems. We provide a brief sketch of the proof for the sake of being self-contained.

Theorem 2.21. *Let $\mathcal{K}_1, \mathcal{K}_2$ be classes of algebras. Then the following hold:*

(1) $\Lambda(\mathcal{K}_1) = \Lambda(\mathcal{K}_2)$ *iff* $Q(\mathcal{K}_1) = Q(\mathcal{K}_2)$
(2) $Th(\mathcal{K}_1) = Th(\mathcal{K}_2)$ *iff* $V(\mathcal{K}_1) = V(\mathcal{K}_2)$

where $Q(\mathcal{K})$, $V(\mathcal{K})$ *denote respectively the quasivariety and the variety generated by* $\mathcal{K}$.

Proof. (1) To prove this condition we need to see that the finite consequence relation of a class of algebras is determined by the quasiequations of that class, and viceversa.

It holds that

$$\gamma_1, ..., \gamma_n \models_{\mathcal{K}} \varphi \quad \text{iff} \quad \bigwedge_{i \leq n} (\gamma_i = \top) \Rightarrow (\varphi = \top) \text{ is valid in } \mathcal{K}$$

for every class of algebras $\mathcal{K}$ so if $Q(\mathcal{K}_1) = Q(\mathcal{K}_2)$, then $\Lambda(\mathcal{K}_1) = \Lambda(\mathcal{K}_2)$.

On the other, given any quasiequation valid in $\mathcal{K}$, then it is expressible in terms of $\Lambda(\mathcal{K})$ because:

$$\bigwedge_{i \in I} (\chi_i = \phi_i) \Rightarrow (\varphi = \psi) \text{ is valid in } \mathcal{K} \quad \text{iff} \quad (\chi_1 \leftrightarrow \phi_1), ..., (\chi_n \leftrightarrow \phi_n) \models_{\mathcal{K}} (\varphi \leftrightarrow \psi)$$

so if $\Lambda(\mathcal{K}_1) = \Lambda(\mathcal{K}_2)$ then $Q(\mathcal{K}_1) = Q(\mathcal{K}_2)$.

(2) As particular cases of the claims in the previous item, it holds that

$$\emptyset \models_{\mathcal{K}} \varphi \quad \text{iff} \quad (\varphi = \top) \text{ is valid in } \mathcal{K}$$

and also

$$(\varphi = \psi) \text{ is valid in } \mathcal{K} \quad \text{iff} \quad \emptyset \models_{\mathcal{K}} (\varphi \leftrightarrow \psi) \quad \square$$

2.4. **Further relevant results on** BL**-logics**

The computer application presented in Chapter 5 is able to cope, on the one hand, with BL (in the sense of working in BL with a fixed set of formulas), and on the other, with all the logics built over finite ordinal sums of BL-chains that are either standard or finitely-valued.

Several technical results are presented in this section. They will be used later for the correctness of the implementation of our solver.

2.4.1. *The treatment of the logic* BL.

Theorem 2.22. *(*[Mon05], [AM03]*)*

The variety of BL*-algebras is generated as a quasivariety by the class of all algebras of the form* $\boxplus_{i \in I} [\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$ *for any finite* I.

Equivalently, the variety of BL*-algebras is generated as a quasivariety by the algebra* $\boxplus_{i \in \mathbb{N}} [\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$.

Corollary 2.23. *(cf. [AG02])*
Given a formula φ and a finite set of formulas Γ,

$$\Gamma \models_{\mathrm{BL}} \varphi \Longleftrightarrow \Gamma \models_{(n+1)[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}} \varphi$$

where n is the number of different variables in $\Gamma \cup \{\varphi\}$, and by $(n+1)[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ we denote the BL*-algebra* $\bigoplus_{i\in\{0,\ldots,n\}} [\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$.

With this corollary, the validity of some formula φ in the logic BL is reduced to validity in the logic defined from the algebra of $(n+1)$ copies of $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$, where n is the number of variables in φ.

2.4.2. *Finiteness on Infinite Valued Fuzzy Logics.* Following the results presented by Aguzzoli et. al. in [AC00, Agu04], we give here a brief selection on results that allow the reducibility of the decision problem on infinite valued Łukasiewicz, Gödel and BL logics to suitable finite valued logics.

Theorem 2.24. *$Ł \models \varphi \Leftrightarrow (Ł_k \models \varphi$ for all $k \leq (\frac{|\varphi|}{n})^n+1) \Leftrightarrow (Ł_{2^{|\varphi|-1}+1} \models \varphi)$ where n is the number of variables in φ and by $|\ \varphi\ |$ we mean the total number of occurrences of propositional variables in φ.*

Theorem 2.25. *$BL \models \varphi \Leftrightarrow ((n+1)Ł_k \models \varphi$ for all $k \leq (\frac{|\varphi|}{n})^n + 1)$ where $(n+1)Ł_k$ denotes the ordinal sum of $n+1$ copies of $Ł_k$.*

On the other hand, it is well known (cf. [Háj98, Th.4.2.18]) that

Theorem 2.26. *$G \models \varphi \Leftrightarrow (G_k \models \varphi$ for all $k \leq n+2) \Leftrightarrow (G_{n+2} \models \varphi)$, where n is the number of variables in φ.*

2.4.3. *An alternative ordinal sum representation.* To work with any BL-chain, we know thanks to the following well known theorem that it is enough to deal with ordinal sums of three concrete ones, $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$, $[\mathbf{0},\mathbf{1}]_{\mathbf{G}}$ and $[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$.

Theorem 2.27. *(Mostert and Shields [MS57], cf. [Háj98]). Any standard* BL*-chain is isomorphic to an ordinal sum ($\bigoplus$) of the $[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$, $[\mathbf{0},\mathbf{1}]_{\mathbf{G}}$ and $[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ standard* BL*-chains.*

Our new aim is to introduce a finitary way (through words) to describe some BL-chains that are ordinal sums.

The words we use for this only involve letters, numbers and the '+' character.

Definition 2.28.

(1) We define the **alphabet** as

$$\mathfrak{L} \coloneqq \{\texttt{1},\texttt{g},\texttt{p},\texttt{p1}\} \cup \{\texttt{l2},\texttt{l3},\texttt{l4}\ldots\} \cup \{\texttt{g2},\texttt{g3},\texttt{g4}\ldots\}$$

A **word w** over a language $L \subseteq \mathfrak{L}$ is a sequence of characters of the form

$$c_0 + c_1 + ... + c_n$$

where for each $i \geq 1$, $c_i \in L$ and $+$ is a literal symbol.
We define the **vocabulary of** $L \subseteq \mathfrak{L}$ as the set $W_L := \{\text{w} : \text{w is a word over } L\}$

(2) The BL algebra associated to a word w=$c_0 + ... + c_n$ is defined as

$$\mathbf{Alg}(\text{w}) := \boxplus_{i \in \{0,...,n\}} \mathbf{Alg}(\mathbf{c_i})$$

where, for each letter, we have

$$\mathbf{Alg}(\mathbf{c_i}) := \begin{cases} \mathbf{2} \oplus (\mathbf{0},\mathbf{1})_{\mathbf{\Pi}} & \text{if } c_i = \texttt{p}, \\ \mathbf{2} \oplus \mathbf{Z}^- & \text{if } c_i = \texttt{p1}, \\ [\mathbf{0},\mathbf{1}]_{\mathbf{Ł}} & \text{if } c_i = \texttt{l}, \\ [\mathbf{0},\mathbf{1}]_{\mathbf{G}} & \text{if } c_i = \texttt{g}, \\ \mathbf{Ł_k} & \text{if } c_i = \texttt{lk}, \\ \mathbf{G_k} & \text{if } c_i = \texttt{gk}. \end{cases}$$

Notice that $\mathbf{2} \oplus (\mathbf{0},\mathbf{1})_{\mathbf{\Pi}} \cong [\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ and $\mathbf{2} \oplus \mathbf{Z}^- \cong \mathbf{Z}^-_{\bullet}$.

Remark 2.29. It is easy to see that for any word w $\mathbf{Alg}$(w) is a complete BL-chain in the sense that supremum and infimum of arbitrary sets of elements can be considered (since $\mathbf{Alg}$(w) is a finite ordinal sum of complete BL-chains).

In the solver, internally, product components will be computed intuitively exploiting Theorem 2.32 and thus all components of type p will be treated like the p1 type[1]. To prove the logic obtained this way coincides with the one that the user assumes obtained by the original BL-chains ordinal sum, a set of theoretical tools is needed.

We proceed to provide a set of important results from algebraic logic that will allow to formalize and prove this idea.

The following are well known results about BL-algebras concerning varieties, quasivarieties and their subdirectly irreducible elements.

Theorem 2.30. *(cf. [BS00])*

*(1) (**Birkhoff**)*

$$\mathcal{V}_{(S.I)} = \mathcal{V}'_{(S.I)} \Rightarrow \mathcal{V} = \mathcal{V}'$$

where $\mathcal{V}_{(S.I)}$ denotes the subdirectly irreductible algebras from the variety $\mathcal{V}$.

[1]The main reason for this lies on performance issues, see Chapter 5

(2) ***Jónsson's Lemma***: *Let $\mathcal{K}$ be a class of algebras, and $V(K)$ be a congruence-distributive variety. If $\mathbf{A}$ is a subdirectly irreducible algebra in $V(\mathcal{K})$ ($\mathbf{A} \in V(\mathcal{K})_{(S.I)}$). Then*

$$\mathbf{A} \in \mathcal{HSP}_u(K)$$

And hence

$$V(K) = \mathcal{IP}_S\mathcal{HSP}_U(K).$$

The next result will allow us to use varieties instead of quasivarieties when working with the algebras associated to words.

Theorem 2.31. *(cf.* [BEGR11, Prop.A8]*) Let $\mathbf{A}$ be a complete* BL*-chain. Then $Q(\mathbf{A})$ is a variety.*

It is interesting to notice that, from this result, and from Theorem 2.18, the following result is direct to obtain.

Theorem 2.32.

$$\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}) = \Lambda(\mathbf{Z}^-_\bullet)$$

On the other hand, one of the main results we need is the following, presented as we will use it in [NEGM05, Prop.3] and based on ideas from [AM03].

Theorem 2.33. *(*[AM03]*) Let $\mathbf{A}_0, \mathbf{A}_1, ..., \mathbf{A}_n$ be linearly ordered hoops and assume $\mathbf{A}_0$ is also bounded (i.e. it is a linearly ordered BL-algebra). Then*

$$ISP_u(\mathbf{A}_0 \oplus \mathbf{A}_1 \oplus ... \oplus \mathbf{A}_n) = \bigoplus_{i\in\{0,...,n\}} ISP_u(\mathbf{A_i})$$

where $\bigoplus_{i\in\{0,...,n\}} ISP_u(\mathbf{A_i})$ denotes the class of all algebras of the form $\bigoplus_{i\in\{0,...,n\}} \mathbf{B_i}$ where for $i \in I$, $\mathbf{B_i} \in ISP_u(\mathbf{A_i})$.

Finally, the following theorems will be used in the equivalent representation of the product component in the ordinal sum we give for $\mathbf{Alg}(\mathrm{w})$

Theorem 2.34. *(*[Háj98]*)*

$$\mathcal{P} = V(\mathbf{2} \oplus \mathbf{Z}^-)$$

From here, in [AFM07] the following result is shown:

Theorem 2.35. *If $\mathbf{A}$ is a nontrivial totaly ordered cancellative hoop, then $ISP_u(A) = ISP_u(\mathbf{Z}^-)$*

We are interested in this result because we trivially know that $[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ is isomorphic to $\mathbf{2}\oplus(\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$. Having that $(\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ is a non-trivial totaly ordered cancellative hoop, we can conclude the following result:

Corollary 2.36.

$$ISP_u((\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}) = ISP_u(\mathbf{Z}^-).$$

On this respect, the following result will be of use later:

Lemma 2.37. *Let* $\mathbf{B},\mathbf{B}',\mathbf{A},\mathbf{C}$ *be linearly ordered hoops, and let* $\mathbf{A}$ *be also bounded. If* $ISP_u(\mathbf{B}) = ISP_u(\mathbf{B}')$ *then*

$$ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}\oplus\mathbf{C}) = ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}'\oplus\mathbf{C}).$$

Proof. Let $\mathbf{B},\mathbf{B}',\mathbf{A},\mathbf{C}$ be linearly ordered hoops that meet the assumptions. By Theorem 2.33, $ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}\oplus\mathbf{C}) = ISP_u(\mathbf{A})\oplus ISP_u(\mathbf{2})\oplus ISP_u(\mathbf{B})\oplus ISP_u(\mathbf{C})$. Then, by the condition of $\mathbf{B},\mathbf{B}'$, it follows that $ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}\oplus\mathbf{C}) = ISP_u(\mathbf{A})\oplus ISP_u(\mathbf{2})\oplus ISP_u(\mathbf{B}')\oplus ISP_u(\mathbf{C})$ and again, by Theorem 2.33, $ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}\oplus\mathbf{C}) = ISP_u(\mathbf{A}\oplus\mathbf{2}\oplus\mathbf{B}'\oplus\mathbf{C})$. □

With all these tools, we can finally prove that the representation given in Definition 2.28 really meets the conditions we need to be able to use indistinctly standard product components $[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ specified by `p` or the discrete characterization $\mathbf{Z}^-_\bullet$ given through `p1`.

We can formalize the idea of two words having the same logic through an equivalence relation $\sim$ between words defined as

$$\text{for all w, v words w} \sim \text{v} \quad \text{iff} \quad \Lambda(Alg(\text{w})) = \Lambda(Alg(\text{v}))\ .$$

Definition 2.38.

(1) Given a word w$=c_0+\ldots+c_n$, w $\in W_L$, $L\subseteq\mathfrak{L}$ we define its **simple translation** by

$$\text{w}^{\#} := c_0'+\ldots+c_n'$$

where

$$c_i' := \begin{cases} \texttt{p1} & \text{if } i = min\{j : 0\le j\le n,\ c_j = \texttt{p}\},\\ c_i & \text{otherwise.}\end{cases}$$

Namely, the first occurrence of `p` in w is replaced by `p1` (if there is any occurrence).

(2) Given a word w$=c_0+\ldots+c_n$, w $\in W_L$ where $L\subseteq\mathfrak{L}$ we define its **translation** by

$$\text{w}^{t} := c_0'+\ldots+c_n'$$

where

$$c_i' := \begin{cases} \mathtt{p1} & \text{if } c_i = \mathtt{p}, \\ c_i & \text{otherwise.} \end{cases}$$

The following result shows the simple translation defined above preserves the logics associated to words.

Proposition 2.39. *Let* $\mathrm{w} \in W_L$, $L \subseteq \mathfrak{L}$ *be a word. Then* $\mathrm{w} \sim \mathrm{w}^{\#}$.

Proof. Assume $\mathrm{w} \neq \mathrm{w}^{\#}$. Then, we can consider

$$\begin{aligned} \mathrm{w} &= \mathrm{w}_1 + \mathtt{p} + \mathrm{w}_2 \\ \mathrm{w}^{\#} &= \mathrm{w}_1 + \mathtt{p1} + \mathrm{w}_2 \end{aligned}$$

To see that $\mathrm{w} \sim \mathrm{w}^{\#}$ we must prove that $\Lambda(Alg(\mathrm{w})) = \Lambda(Alg(\mathrm{w}^{\#}))$, namely $Q(Alg(\mathrm{w})) = Q(Alg(\mathrm{w}^{\#}))$.

Now, since the algebra associated to a word is always a complete BL-algebra, by Theorem 2.31 it is enough to prove $V(Alg(\mathrm{w})) = V(Alg(\mathrm{w}^{\#}))$.

$V(Alg(\mathrm{w})) = V(Alg(\mathrm{w}^{\#}))$ will be guaranteed if we see $V(Alg(\mathrm{w}))_{(S.I)} = V(Alg(\mathrm{w}^{\#}))_{(S.I)}$ thanks to Theorem 2.30(2).

Thanks to *Jónsson's Lemma*, we can reduce this statement to checking $HSP_u(Alg(\mathrm{w})) = HSP_u(Alg(\mathrm{w}^{\#}))$.

Since both sides are closed by H, it is enough to see $ISP_u(Alg(\mathrm{w})) = ISP_u(Alg(\mathrm{w}^{\#}))$, i.e. that

$$(1) \qquad ISP_u(Alg(\mathrm{w}_1 + \mathtt{p} + \mathrm{w}_2)) = ISP_u(Alg(\mathrm{w}_1 + \mathtt{p1} + \mathrm{w}_2)).$$

Reached this point of simplification, the rest of the proof will be devoted to prove this last equality.

By definition, (1) means

$$ISP_u(Alg(\mathrm{w}_1) \boxplus Alg(\mathtt{p}) \boxplus Alg(\mathrm{w}_2)) = ISP_u(Alg(\mathrm{w}_1) \boxplus Alg(\mathtt{p1}) \boxplus Alg(\mathrm{w}_2))$$

At this point, we can represent each one of the two t-norms ordinal sums in their respective basic (for the `p`, `p1` components) hoops ordinal sum, i.e. we need to check that

$$ISP_u(Alg(\mathrm{w}_1) \oplus \mathbf{2} \oplus (\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}} \oplus Alg(\mathrm{w}_2)) = ISP_u(Alg(\mathrm{w}_1) \oplus \mathbf{2} \oplus \mathbf{Z}^- \oplus Alg(\mathrm{w}_2))$$

To this aim, it is enough to see that $Alg(\mathrm{w}_1)$, $(\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$, $\mathbf{Z}^-$, and $Alg(\mathrm{w}_2)$ satisfy the requirements of Lemma 2.37. Just consider $\mathbf{A} := Alg(\mathrm{w}_1)$, $\mathbf{C} := Alg(\mathrm{w}_2)$, $\mathbf{B} := (\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}}$ and $\mathbf{B}' := \mathbf{Z}^-$ and remember that $ISP_u((0,1]_{\Pi}) = ISP_u(\mathbf{Z}^-)$ (by Corollary 2.36).

Then, we can conclude that

$$ISP_u(Alg(\mathrm{w}_1) \oplus \mathbf{2} \oplus (\mathbf{0}, \mathbf{1}]_{\mathbf{\Pi}} \oplus Alg(\mathrm{w}_2)) = ISP_u(Alg(\mathrm{w}_1) \oplus \mathbf{2} \oplus \mathbf{Z}^- \oplus Alg(\mathrm{w}_2)),$$

i.e., we have shown condition (1). □

With this result, we can naturally extend the translation operation to the whole word (since it is finite) and formulate the fact that is the main axis of the correctness of the new solver:

Corollary 2.40. *Given a word* $\mathrm{w} \in W_L$, $L \subseteq \mathfrak{L}$,

$$\mathrm{w} \sim \mathrm{w}^t$$

This corollary will allow us to replace any word by an equivalent one defined over $\mathfrak{L} \setminus \{\mathtt{p}\}$.

2.5. **Extensions of** BL

Two different ways to extend the previously presented logics are very interesting to point out: constants and the Baaz's projection operator. Both are natural extensions of the classical fuzzy language and had been widely studied from a theoretical point of view.

We will provide here a brief explanation of these notions and the use the reasoner gives for them.

2.5.1. *Constants.* Expansions of Basic Fuzzy Logic, or indeed of many other systems of many-valued logic, with propositional truth constants have been studied from various points of view.

Pavelka [Pav79a, Pav79b, Pav79c] defined a deductive system allowing for reasoning with degrees of truth via including constants for all truth values into the propositional language, and his completeness result stated, for each theory and each formula, the equality of the provability degree and the validity degree of the formula with respect to the theory.

For the purpose of this report, it is interesting the more specific case presented in [EGGN07]. There, the authors start from a standard BL-algebra $[0,1]_*$ and introduce propositional constants for a countable subalgebra $C \subset [0,1]$, and present theoretical results on these extensions.

In our case, due to the definition (by pairs) of the hoops ordinal sum, and to the internal way of dealing with the Product components in an ordinal sum, the treatment of constants differs from the approach given in [EGGN07]. In this work, constants are not expressed as numbers in $[0,1]$, but rather in a pair format telling also the component in the ordinal sum of the constant, formally:

Definition 2.41. A **constant c** for a word $\mathrm{w} \coloneqq c_0 + c_2 + ... + c_n$ is given as

$$\mathrm{c} \coloneqq \langle v, n \rangle$$

where $\langle v, n\rangle \in \mathbf{Alg}(\mathbf{c_i}) \cap \mathbb{Q}^2$ for some $0 \leq i \leq n$. In particular, this naturally implies that c $\in \mathbf{Alg}$(w).

The interpretation in $\mathbf{Alg}$(w) of each constant will be given by

$$\overline{\mathrm{c}} := c$$

Intuitively, with this definition we are simply allowing rational constants for any of the components of the ordinal sum of $\mathbf{Alg}$(w) (i.e. for each one of the BL-chains associated with each character of the word). Since all the hoops involved are linearly ordered, the intuitive meaning of a formula of the shape c $\rightarrow \varphi$ is $\overline{\varphi} \geq c$ (where $\overline{\varphi}$ is the interpretation of φ in $\mathbf{Alg}$(w)).

As a future work it is planned to extend the reasoner to accept also the usual constants written as numbers in $[0, 1]$. Limits $(0, a_1)(a_2, a_3)...(a_{n-1}, a_n)$ in $[0, 1]$ should be given for each component of the ordinal sum, allowing an adequate use of the constants (similar to the treatment presented in [EGGN07]). This will be feasible whenever no discrete product component is involved.

2.5.2. *Baaz's projection.* The Baaz Delta operator Δ is a "defuzzification" operator that can be added to any BL-algebra $\mathbf{A}$ and which is defined as

$$\Delta x := \begin{cases} \top & \text{if } x = \top, \\ \bot & \text{otherwise.} \end{cases}$$

Due to lack of time, the inclusion of the Baaz projector operator as part the language supported by the solver has not been implemented. However, the use of constants and the specification of equations in the solver allows the same expressive power. With this we mean that given a BL-algebra $\mathbf{A}$,

$$\Delta(\varphi) \in Th(\mathbf{A}) \text{ iff } \varphi = \top \text{ is valid in } \mathbf{A}.$$

It is noticeable that all the theoretical results we presented on Section 2.4 keep holding when the Baaz Projector is added (see for instance [Mon01]). It is proposed as a future work to add this functionality to the reasoner.

3. Complexity Issues

The present chapter is devoted to an analysis of computational complexity of the propositional logics we have presented in Chapter 2. First we will summarize necessary material about the theory of polynomial complexity and arithmetical hierarchy. In a second section we will present the main results on complexity about Łukasiewicz Gödel and Product Logics, and all the logics built over BL-chains.

3.1. Computational Complexity

We will assume the reader is familiar with the concept of a *Turing Machine*. We will work with decidable languages, i.e. by definition, the ones for which there is a Turing Machine that for any input it always halts.

Definition 3.1. Given M a Turing Machine, the language accepted by M is the set of all words w for which the machine stops and it is in the accepting internal state.

A Turing machine is deterministic whenever for any pair $\langle$*state*, *input symbol*$\rangle$ there exists at most one possible execution. If that is not the case, the machine is said to be non-deterministic.

A machine M runs in polynomial time if there is a polynomial $p(u)$ such that for each word w of length n, each computation with the input w stops after $\leq p(n)$ steps.

With these concepts in mind, we can proceed to define two classes of complexity we are interested in:

Definition 3.2.

(1) **P** is the class of all languages accepted by a deterministic Turing Machine on polynomial time.
(2) **NP** is the class of all languages accepted by a non-deterministic Turing Machine on polynomial time.
(3) A language is in class **coNP** if its complement is in **NP**.
(4) A language L is **NP-complete** if it is in **NP** and for each L' in **NP** exists a function f computed by a deterministic Turing machine running on polynomial time such that for each word w' in the alphabet of L', $w' \in L'$ iff $f(w') \in L$. The definition for **coNP-complete** is analogous.

In classical propositional logic, the sets

$$\begin{aligned} SAT &:= \{\varphi : \exists\ evaluation\ v,\ v(\varphi) = \top\} \\ TAUT &:= \{\varphi : \forall\ evaluation\ v,\ v(\varphi) = \top\} \end{aligned}$$

determine the main problems, and because of this reason, they are the ones that have been studied for determining the complexity of the logic. It is well known (*Cook's theorem*) that the set SAT is **NP-complete** and so $TAUT$ is **coNP-complete**.

3.2. **Complexity in a many valued frame**

With a many-valued logic, the definition of the SAT and $TAUT$ problems is richer, as the classical dichotomy no longer applies. For a fixed semantics given by an algebra $\mathbf{A}$, we define the following sets of formulas (cf. [Háj98]):

$$\begin{aligned} SAT^{\mathbf{A}}_{\top} &:= \{\varphi : \exists\ \mathbf{A} - evaluation\ v,\ v(\varphi) = \top\} \\ SAT^{\mathbf{A}}_{pos} &:= \{\varphi : \exists\ \mathbf{A} - evaluation\ v,\ v(\varphi) > \bot\} \\ TAUT^{\mathbf{A}}_{\top} &:= \{\varphi : \forall\ \mathbf{A} - evaluation\ v,\ v(\varphi) = \top\} \\ TAUT^{\mathbf{A}}_{pos} &:= \{\varphi : \forall\ \mathbf{A} - evaluation\ v,\ v(\varphi) > \bot\} \end{aligned}$$

These sets are referred to as $\top$-satisfiable, positively satisfiable, $\top$-tautologies, and positive tautologies.

The generalization of them to a class of algebras is the natural way, i.e. given $\mathcal{K}$ a class of algebras,

$$\begin{aligned} SAT^{\mathcal{K}}_{\top} &:= \bigcup_{\mathbf{A}\in\mathcal{K}} SAT^{\mathbf{A}}_{\top} \\ SAT^{\mathcal{K}}_{pos} &:= \bigcup_{\mathbf{A}\in\mathcal{K}} SAT^{\mathbf{A}}_{pos} \\ TAUT^{\mathcal{K}}_{\top} &:= \bigcap_{\mathbf{A}\in\mathcal{K}} TAUT^{\mathbf{A}}_{\top} \\ TAUT^{\mathcal{K}}_{pos} &:= \bigcap_{\mathbf{A}\in\mathcal{K}} TAUT^{\mathbf{A}}_{pos} \end{aligned}$$

Notice than unlike in classical logic, for many-valued semantics there is no simple relationship between its $TAUT$ and SAT problems.

We will provide here a sum up of the main points in complexity issues for BL and its principal extensions, referring for all the following results to [AGH05] and [CH09].

3.2.1. *Lukasiewicz logic.*

Theorem 3.3. *([Mun87], cf. [Háj98]) Consider for $\theta \in [0,1]$,*

$$SAT^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}}_{\theta} := \{\varphi : \exists x \in [0,1]^n,\ \varphi^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}}(x) \geq \theta\}.$$

Then
$SAT^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}}_{\theta}$ *is* **NP-*complete*** *for each $\theta \in [0,1] \cap \mathbb{Q}$.*

In particular, if we consider $\theta = \top$, we get

$$SAT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}} \quad \textit{is} \quad \textbf{NP-}\textbf{\textit{complete}},$$
$$SAT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}} \quad \textit{is} \quad \textbf{NP-}\textbf{\textit{complete}}.$$

And considering $SAT_0^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}} \setminus SAT_T^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}}$ we have

$$TAUT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}} \quad \textit{is} \quad \textbf{coNP-}\textbf{\textit{complete}},$$
$$TAUT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}} \quad \textit{is} \quad \textbf{coNP-}\textbf{\textit{complete}}.$$

Corollary 3.4. *$\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{L}})$ and $Th([\mathbf{0},\mathbf{1}]_{\mathbf{L}})$ are* **coNP-*complete***.

3.2.2. *Gödel and Product logic.* We present here the results shown in [Háj98].

Theorem 3.5. *$SAT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{G}}} = SAT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{G}}} = SAT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}} = SAT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}}$ and all these sets are equal to classical SAT (and thus* **NP-*complete***).*

Theorem 3.6. *$TAUT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{G}}} = TAUT_{pos}^{[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}} = TAUT$ (classical), and so are* **coNP-*complete***.*

Theorem 3.7. *$TAUT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{G}}}$ and $TAUT_{\top}^{[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}}$ are* **coNP-*complete***.*

Corollary 3.8.

(1) $\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{G}})$ and $Th([\mathbf{0},\mathbf{1}]_{\mathbf{G}})$ are **coNP-*complete***.
(2) $\Lambda([\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}})$ and $Th([\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}})$ are **coNP-*complete***.

3.2.3. *Finitely-valued logics.*

Theorem 3.9. *$SAT_{pos}^{\mathbf{L_n}}$, $SAT_{\top}^{\mathbf{L_n}}$, $SAT_{\top}^{\mathbf{G_n}}$ and $SAT_{pos}^{\mathbf{G_n}}$ are in* **NP**.
$TAUT_{pos}^{\mathbf{L_n}}$, $TAUT_{\top}^{\mathbf{L_n}}$, $TAUT_{\top}^{\mathbf{G_n}}$ and $TAUT_{pos}^{\mathbf{G_n}}$ are in **coNP**.

Corollary 3.10. *$\Lambda(\mathbf{L_n})$, $\Lambda(\mathbf{G_n})$, $Th(\mathbf{L_n})$ and $Th(\mathbf{G_n})$ are in* **coNP**

3.2.4. *Basic Logic.* The results on the complexity of BL are proved using ordinal-sum decomposition of standard BL-algebras into $[\mathbf{0},\mathbf{1}]_{\mathbf{L}}$, $[\mathbf{0},\mathbf{1}]_{\mathbf{G}}$ and $[\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}$ BL-chains. The following result is based on the complexity for each one of the previously commented logics.

Theorem 3.11. *([BHMV02]) $\Lambda(\mathcal{BL})$ and $Th(\mathcal{BL})$ are* **coNP-*complete***.

3.2.5. BL-*chains.* Using all the previous results, a more general theorem is obtained for standard BL-chains. Thanks to the completeness of BL with respect to this class ($\mathcal{BL}sc$), this result closes the complexity issues that concern the logics we work with.

We will use the notation

$$SAT_1^* \coloneqq \bigcup_{\mathbf{A}\in\mathcal{BL}st} SAT_\top^{\mathbf{A}}$$
$$TAUT_1^* \coloneqq \bigcap_{\mathbf{A}\in\mathcal{BL}st} TAUT_\top^{\mathbf{A}}$$

for denoting the $\top$-satisfiable formulas and the and $\top$-tautologies of all the standard BL-chains, determined by a continuous t-norm $*$.

Theorem 3.12. *([AGH05], [BHMV02])*

(1) SAT_1^* *is* **NP-*complete*.**

(2) $TAUT_1^*$ *is* **coNP-*complete*.**

Corollary 3.13. *For any standard* BL*-chain* $\mathbf{A}$, $\Lambda(\mathbf{A})$ *and* $Th(\mathbf{A})$ *are* **coNP-*complete*.**

4. Satisfiability Modulo Theories

Intuitively, Satisfiability Modulo Theories (SMT) can be seen as generalization of the SAT problem by adding the ability to handle arithmetic and other theories. In this work it is an interesting field to be detailed because after we found studies concerning the adequacy of SMT solvers for solving Constraint Satisfaction Problems ([ABMV12]), we considered these solvers to be of great use to work with the Basic Logic and its axiomatic extensions simply by interpreting in them the constraints imposed by these logics.

In this chapter a formal description of Satisfiability Modulo Theories (SMT) concept, design and implementations is provided.

4.1. **The Satisfiability Problem**

Satisfiability, namely the problem of determining if a formula (in real applications, this formula is expressing a constraint) has a solution in a certain logic, is one of the most fundamental problems in theoretical computer science.

Constraint satisfaction problems arise in many diverse areas including graph and game theory problems, planning, scheduling, software and hardware verification, extended static checking, optimization, test case generation or type inference. The most well-known constraint satisfaction problem is propositional satisfiability SAT, where the logic checked is the classical propositional logic: decide whether a formula using classical logical connectives can be made true by choosing true/-false values for its variables.

Many of these constraint satisfaction problems can be encoded by Boolean formulas and solved using Boolean satisfiability (SAT) solvers. However, other problems require the added expressiveness of equality, uninterpreted function symbols, arithmetic, arrays, datatype operations or quantifiers. For example, many applications of formal methods that rely on generating First-Order formulas over theories of the real numbers or integers (including fuzzy logics) are in need of more expressive logical languages and solvers.

Thus, a formalism extending SAT called Satisfiability Modulo Theories (SMT), has also been developed to deal with these more general decision problems. An SMT instance can informally be understood as a first order Boolean formula in which some propositional variables are replaced by predicates with predefined interpretations from background theories. Namely, these predicates are binary-valued functions over non-binary variables.

In Chapter 3 we have remarked that, for instance, the complexity of Hájek's Basic Logic (and its extensions) is **coNP-complete**. This is not an exception, as many of the Constraint Satisfaction Problems reach at least the **NP** complexity class. Due to this high computational complexity and to the fact that real problems are not interested in validity in general, but with respect to a fixed background theory, the idea is not to build a procedure that can solve arbitrary SMT problems, but to focus on specialized SMT solvers.

As done with efficient *SAT* solvers, when working with concrete problems, the procedures can be highly simplified and fastened paying attention to implementation details. In recent years, there has been an enormous progress in the scale of problems that can be solved, thanks to innovations in core algorithms, data structures, heuristics, and other methods, and for example, modern *SAT* procedures can check formulas with hundreds of thousands variables and millions of clauses.

In the case of SMT, similar progress has been observed in the procedures for the more commonly occurring theories that not only work with FOL but also in fragments of it (for instance, quantifier free formulas). For many of these, specialized methods actually yield decision procedures for the validity ground formulas or some subset of them. This is for instance the case, thanks to classical results in mathematics, for the theory of real numbers or the theory of integer numbers (for formulas with no multiplication symbols). In the last two decades however, specialized decision procedures have also been discovered for a long, and still growing, list of theories of other important data types such as certain theories of arrays and of strings, variants of the theory of finite sets, the theory of several classes of lattices, the theories of finite, regular and infinite trees, and the theory of lists, tuples, records, queues, hash tables, and bit vectors of a fixed or arbitrary finite size.

4.2. **Formal Definitions**

Based on pioneer works proposing the use of SMT solvers in formal methods in the 80s [NO80, Sho81, BM79], on the last ten years we have lived an increasing interest on this field, and research on the foundational and practical aspects of SMT has rapidly grown. Several SMT solvers have been developed in academia and industry with continually increasing scope and performance. Out of interest, we can cite here examples integrated into interactive theorem provers for high-order logic (such as HOL and Isabelle), extended static checkers (such as CAsCaDE, Boogie, and ESC/Java 2), verification systems (such as ACL2, Caduceus, SAL, UCLID and Why) or model checkers (such as

BLAST) among others. In industry, as important centers with SMT-related projects we can name Microsoft Research, Cadence Berkeley Labs, Intel Strategic CAD Labs and NEC Labs.

Most approaches for automated deduction tools rely on case-analysis for its core system. In the case of SMT, most of the solvers exploit SAT procedures for performing case-analysis efficiently.

In this section basic techniques used in state-of-the-art SAT solvers and the more common approaches to the SMT problem are detailed.

4.2.1. *SAT encodings.* Most state-of-the-art SAT solvers (Glucose [AS09], Minisat [ES03], BerkMin [GN07]) today are based on Conflict-Driven Clause Learning algorithm (CDCL), originally grown from the Davis-Putnam-Logemann-Loveland (DPLL) procedure [DP60, DLL62].

The DPLL algorithm is is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem. The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the splitting rule, as it splits the problem into two simpler sub-problems. The simplification step essentially removes all clauses which become true under the assignment from the formula, and all literals that become false from the remaining clauses.

The great improvements in the performance of DPLL-based SAT solvers achieved in the last years are due, on the one hand, to better implementation techniques, and on the other, to several conceptual enhancements on the original DPLL procedure, aimed at reducing the amount of explored search space, such as backjumping (a form of non-chronological backtracking), conflict-driven lemma learning, and restarts. On the other hand, correctness has been proved for each of these methods, ensuring the coherence of their usage.

These advances make it now possible to decide the satisfiability of very complex *SAT* problems.

Detailed description of existing procedures is out of the scope of this work, but a uniform, declarative framework for describing DPLL-based solvers, *Abstract DPLL*, can be found in [NOT06].

4.2.2. *SMT-solvers approach.* Following the more recent SMT literature, given a signature Σ, we define a theory T over Σ as just one or more (possibly infinitely many) Σ-models. Then, a ground Σ-formula φ is satisfiable in a Σ-theory or is T-satisfiable) if and only if there is

an element of the set T that satisfies φ. Similarly, a set Γ of ground Σ-formulas T-entails a ground formula φ ($\Gamma \models_T \varphi$) if and only if every model of T that satisfies all formulas in Γ satisfies φ as well.

We say the satisfiability problem for theory T is decidable if there is a procedure Υ that checks whether any ground (and hence, quantifier free) formula is satisfiable or not. In this case, we say Υ is a decision procedure for T or a T-solver.

The proof of correctness of SMT methods for decidable background theories is a step that has to be checked for each solver. Satisfiability procedures must be proved sound and complete; while soundness is usually easy, completeness requires specific model construction arguments showing that, whenever the procedure finds a formula satisfiable, a satisfying theory interpretation for it does indeed exist. This means that each new procedure in principle requires a new completeness proof.

Working with a decidable theory T, there are two main approaches for determining the satisfiability of a formula with respect to T, each with its own pros and cons: *eager* and *lazy*.

Eager SMT approach:
This approach consists on ad-hoc translations from an input formula and relevant data from the theory T into a set of equisatisfiable propositional formulas (see for instance [SSB02] for this procedure), which is then checked by a SAT solver for satisfiability.

The good point of this approach is that it can always make use of the lasts existing SAT solvers, overcoming the problem of relatively big translation results.

However, the problem lies on the exponential cost of the translation operation for making the problem treatable by a SAT solver. For this, sophisticated ad-hoc translations are needed for each theory, and experiments have shown the explosion on needed resources when escalating the problems (see [dMR04]). Also, it has been studied the difficulty of combining several theories. To address these issues, the latest research on SAT encodings focuses on general frameworks that allow incremental translations and calls to the SAT solver, and general mechanisms for combining encodings for different theories.

From a theoretical point of view, proving soundness and completeness is relatively simple because it reduces to proving that the translation is satisfiability invariant (but that proof needs to be done for each defined translation).

Lazy SMT approach:

Instead of an ad-hoc translation for each theory into a SAT problem, a specialized T-solver for deciding the satisfiability of conjunctions of theory literals can be defined. Then, the objective is combining the strength of this solver with the existing SAT-solvers to produce an efficient SMT-solver.

A lot of research has been done in the matter of combining these two solvers. The most widely used approach in the last few years is usually referred to as the *lazy* approach [dMR02, BCLZ04, ACGM05].

The idea behind this approach is such that each atom occurring in a formula φ to be checked for satisfiability is initially considered simply as a propositional symbol, not taking into account the theory T. Then, the formula is processed by a SAT solver, which determines its propositional satisfiability. If φ was found unsatisfiable by the SAT solver, then it also is T-unsatisfiable. In other cases, the SAT solver returned a propositional model M of φ, and then this assignment will be checked by a T-solver. If M is found T-consistent, it is a T-model of φ. Otherwise, the T-solver generates a ground clause rejecting that assignment. This formula is then added to φ by propositional conjunction, and the SAT solver is started again. This process is repeated until a T-model is found or the SAT solver returns the formula is unsatisfiable.

For details on satisfiability in first-order theories and results on combining several theories for working under the *lazy approach* see for instance [dMDS07].

Most of the currently implemented solvers follow this approach, and include a high number of available theories like linear, difference and non-linear arithmetics, bit-vectors, arrays and free functions among others. By modularity and correctness of the SAT solvers, the correctness for each particular theory is the main point proving the correctness of the solver (up to exactness of the implementation).

A general definition for SMT has been studied in [NOT06] providing the $DPLL(T)$ approach, a general modular architecture based on a general DPLL engine parametrized by a solver for a theory T of interest. Here details about the theories used in our particular case will be given. In the implementation of the new solver a combination of the linear arithmetic and the array theories is used, as it will be detailed in Section 4.2

- **Linear arithmetic**
  Linear arithmetic (LA) constraints have the form $c_0 + \Sigma_{i=1}^{n} c_i \cdot x_i \leq 0$, where each c_i for $0 \leq i \leq n$ is a rational constant and the variables x_i range over $\mathbb{R}$. The LA-solver algorithm implemented by the SMT solver used for the experiments, z3

[dMB08], is based on the method proposed by de Moura and Dutertre in [DdM06].

- **Arrays**
  This theory was introduced by McCarthy in [McC62], and its functions are reduced to *read (a, i, v)* and *write(a, i, v)*. Depending on the theory over which the arrays are used, and the dimension specified to the array in its creation, a and i will differ on type. In our case, the implementation of the solver is built over arrays of dimension two of real values, so operations are defined as follows:

$$\begin{aligned} write &: \mathbb{R}^2 \times \mathbb{R} \times \{0,1\} \longrightarrow \mathbb{R}^2 \\ read &: \mathbb{R}^2 \times \{0,1\} \longrightarrow \mathbb{R} \end{aligned}$$

  with

$$write([a,b],c,i) := \begin{cases} [c,b] & \text{if } i = 0, \\ [a,c] & \text{if } i = 1. \end{cases}$$

$$read([a,b],i) := \begin{cases} a & \text{if } i = 0, \\ b & \text{if } i = 1. \end{cases}$$

4.3. **Standardisation: Language and Solvers**

It seems natural that for different SMT-solvers, given their specific treatment of problems and so their different working methods, different interfaces and input formats are given. Until 2002, no standardization existed, and so comparing different SMT solvers was a difficult task.

To reduce this drawback, the SMT community launched in 2002 the SMT-LIB initiative [BST10] which is currently backed by the vast majority of research groups in SMT. SMT-LIB defines standard input/output formats and interfaces for SMT solvers and also provides an on-line repository of benchmarks for several theories.

The standardisation led to the creation of a an annual competition for SMT-solvers, SMT-COMP [SC], where state of the art SMT solvers show their strengths in different kind of tests. Here was searched for deciding the more interesting SMT-solver to test our results (see section 5.4). The latest results at SMT-COMP (2011), where Z3[dMB08] solver from Microsoft Research obtained the best results for the theories interesting for the final solver made us select Z3 for the experiments.

5. The New Solver

The main point of this research is the presentation of a software application that works as a logical reasoner for the logic BL and a wide family of its extensions.

In this chapter a deep description of the new solver is given. It is provided a complete theoretical justification of the correctness of its design based on results presented on Chapter 2, and a detailed description of the implementation methods is given for the interested reader. Secondly, a user manual and several examples of use are provided for simplicity on the use of the solver. Finally, a section explaining the performed tests and the efficiency results obtained is shown. As an short section, we have included here information about a complementary solver that is interesting from a theoretical point of view but whose slow performance leads it to be a non practical application.

5.1. **What does it do?**

The software application presented here performs, over a given logic, either the task of checking if a certain formula is a theorem (valid for all possible evaluations), or if a pair formed by a sequence of premises and a formula hold in the logical consequence relation, or if a certain set of equations is satisfiable (and how). Explicitly, as a concise presentation of the solver, its use can be resumed in terms of input/output through the following lines:

- A **logic** L shall be given in the input, either `bl` either as a word w over the language $L := \mathfrak{L}\backslash\{\texttt{p1}\}$ (see Definition 2.28), i.e. as a chain of characters of the form `"[`c_1`+`c_2`+...+`c_n`]"` where each c_i is comprehended in {`l,g,l2,l3...,g2,g3...,p`} and + is a literal symbol.

  In this last case, the solver will assume the logic to work with is the one obtained from **Alg**(w), where w is the input word.
- An **output file** will be given which contains the intermediate SMT-LIB codes.
- A **task** must be specified, and this will determine other inputs that shall be given and the output of the solver. A brief specification is as follows:
  - If the task is fixed to the theoremhood prover option (`th`), a formula shall be given. Then, after the execution, the program will show a message confirming if the formula is or is not a theorem in the logic L.
  - If the task is fixed to the logical consequence option (`c`), a formula and a set of formulas (premises) shall be given.

Then, after the execution, the program will show a message confirming if the formula is or is not a logical consequence of the premises set in the logic L.

– If the task is fixed to the satisfiability checking option (`s`), a set of equations shall be given. Then, after the execution, the program will show a message confirming if the set of equations is satisfiable or not in the logic L, and if it is, it will show an assignment of the variables that satisfies it.

5.2. **Theoretical basis**

Inspired by the approach of Ansótegui et. al in [ABMV12], we conceived the idea of programming a more general solver for fuzzy logics using an SMT solver, exploiting theoretical results about these logics.

In their work, Ansótegui et. al. describe a theorem prover software capable of determining whether a formula φ of the Łukasiewicz infinitely-valued logic is a tautology. They do so developing a satisfiability checker for Łukasiewicz logic, and then asking whether there is an interpretation I such that $I(\varphi) < 1$. If such interpretation does not exist, then φ is a tautology. In their work they present the Gödel logic solver too, but the approach to solve product logic was not successful from a practical point of view because of efficiency matters.

In this work, a generalization of this approach is proposed, exploiting results presented on Chapter 2 to expand the solver tasks and to enhance its performance. The work presented is a logical reasoner that allows the specification, in term of its components, of any continuous t-norm and the use of BL too. Also is allowed to work with finitely-valued Łukasiewicz and Gödel logics, and as an important feature, all these logics can be used extended with rational truth-constants. Also, it seemed interesting to add to the software more flexibility in the sense of performing more tasks than just testing the theoremhood of a formula in a certain logic. In the new solver, checking whether a given formula (possibly with truth-constants) is a logical consequence of a finite set of formulas (possibly with truth-constants as well) is feasible, and also is allowed to ask about the models satisfying a certain set of equations.

The main idea for doing this comes from Theorem 2.27, which asserts that any continuous t-norm v can be expressed as the ordinal sum of the three main continuous t-norms $*_{\mathrm{L}}$, $*_G$ and $*_{\Pi}$. From this result, defining these three t-norms and the ordinal sum within the SMT-LIB

language would provide not only a theoremhood prover for any continuous t-norm based fuzzy logic (included infinite ordinal sums), but also an application for checking finite logical consequence and a model generator tool for these logics (very useful for looking for counterexamples).

The BL case was not considered in [ABMV12] because the semantics is based on a family of continuous t-norms, and not just on one. However, by Theorem 2.23 we can reduce proofs over BL, when working with concrete formulas, to proofs over the logic defined with t-norm given by an ordinal sum of $(n+1)$ Łukasiewicz components, where n is the number of different variables in the set of formulas involved. This will be the method we use in our solver for the implementation of BL, since it has performance advantages over using the idea in Theorem 2.25.

To implement the ordinal sum as defined above, as done in [ABMV12], we initially considered the classical definition of the three t-norms, i.e. the ones from Definitions 2.9, 2.10 and 2.11 respectively. These definitions are given using only addition and multiplication over the real unit interval (in the case of the finitely-valued Łukasiewicz and Gödel logics, over a corresponding subset of the natural numbers). Thus, it is natural to implement a solver for these three kind of logics and all the family over them using the QF_LRA and QF_NLRA theories of the SMT solver, as it was done for each particular case in [ABMV12]. For Łukasiewicz and Gödel t-norms, a lineal real arithmetic theory is enough to define the needed operations, and whenever the product t-norm is involved in the decomposition, the non-linear real arithmetic is used to deal with Product t-norm and residuum. The problem with this approach is that, computationally, product and division are very slow operations, so when the Product t-norm is involved on the definition of the t-norm of the logic, the efficiency of the solver falls dramatically, even for really simple cases, as we will describe on Section 5.4.

For this reason, for the particular case of Product Logic a new methodology has been designed. To overcome the problems inherited of the non-linearity in $[0,1]$ an alternative coding based on QF_LIA has been defined. This can be partially done thanks to Theorem 2.32, that shows that the variety of Product algebras is also generated by a discrete linear product algebra: the one generated by $\mathbf{Z}^-_\bullet$. Also, we have proven that the operation of ordinal sum is well behaved (in the sense it preserves the logic) when we do this exchange (see Corollary 2.40). Therefore, for dealing with Product logic it is enough to deal with this discrete algebra ($\mathbf{Z}^-_\bullet$); and this particular algebra can be codified using just natural numbers and addition over the negative natural

numbers with a minimum (i.e., Presburger Arithmetic). In Section 5.4, the enhancement will be detailed.

Finally, from a pure programming point of view, and for simpler definition of our program, the notion of Ordinal Sum for hoops given in Definition 2.14 is exploited. Also, the usage of pairs of values for each variable has been conceived (see the definition of the ordinal sum for hoops 2.14 for details), assigning a real number for the value inside the component and a natural number to identify the component, allowing the implementation of the general hoop ordinal sum and providing naturally the option of specifying ordinal sums of infinite and finite number of components.

The formal definition of the input of the logic for the application is directly based on the Definition 2.28 from Section 2.4.3.

Indeed, for working with the sum t-norm, we can semantically work with the sum of the correspondent hoops. Moreover, thanks to Corollary 2.40, we can exclusively work with discrete product component which is much more efficient.

Formally, the reasoner takes as input a word over the language $L = \mathfrak{L} \setminus \{\texttt{p1}\}$, i.e., $\mathrm{w} = c_0 + c_1 + ... + c_n$ where for each $0 \leq i \leq n$, $c_i \in \{\texttt{l,g,p}\} \cup \{\texttt{l2,l3...}\} \cup \{\texttt{g2,g3...}\}$.

With this, the input w corresponds to is the logic generated by the BL-chain formed by the ordinal sum of each of the BL-chains associated with each character. Namely,

$$Logic_{user} = \Lambda(\mathbf{Alg}(\mathrm{w})),$$

see Definition 2.28 for detais on $\mathbf{Alg}(\mathrm{w})$.

Internally, the reasoner will work with the logic generated by the BL-chain formed with the hoops ordinal sum of the hoops specified by each component in the translation of w (see Definition 2.38), where a component of type `p` is changed by a component of type `p1` which can be discretely computed. Namely,

$$Logic_{prog} = \Lambda(\mathbf{Alg}(\mathrm{w}^t)).$$

We saw on Corollary 2.40 that these two logics ($\Lambda(\mathbf{Alg}(\mathrm{w}))$ and $\Lambda(\mathbf{Alg}(\mathrm{w}^t))$) coincide, so the design of the software using this approach is correct from a theoretical point of view. The answer the user expects is the one shown, but it is worth saying that using $\Lambda(\mathbf{Alg}(\mathrm{w}^t))$ instead of $\Lambda(\mathbf{Alg}(\mathrm{w}))$ leads to a cleaner and simpler programming and to a much faster execution.

In the solver, the above definitions are directly translated into z3 code, and with that results the SMT-solver will produce the solution of the given problem.

5.3. **Usage**

By now, the utilization of our solver is limited to terminal (command line). This allows a simpler codification and a more direct access to the options of the solver. As a future work, the development of a graphical interface is desired, to make the application more accessible and user-friendly.

5.3.1. *Pre execution.* The program is implemented in python and z3, and it is meant to be used from the terminal. To run it, it is necessary to have python and the z3 solver ([z3w]) installed in the computer.

Before its use, a configuration file should be changed according to each user. Since the SMT solver is internally used, in file `configuration.py` the line

```
Z3_LOCATION = "/.../z3"
```

must be modified to meet the user's Z3 folder. It shall have the relative (to the reasoner main folder) or the absolute path in the user's computer to the z3-solver general folder, obtained after downloading and decompressing the z3-solver ([z3w]).

5.3.2. *Inputs.* From the main solver, the application is called as

```
>python ctnormsolver.py
```

It needs a certain amount of parameters to work. On the one hand, for easiness of use, a help option has been included, providing a brief description of the attributes to be used. When used, the output will be the following:

Listing 1. help

```
>python ctnormsolver.py -h
usage: python fuzzy.py [-h] [-l L] [-t T] [-f F] [-p P]
    [-eq EQ] [-out OUT]

BL and extension logics reasoner.

optional arguments:
  -h, --help  show this help message and exit
  -l L        'bl' or a list '[a+b+...]' of components
                -logics- in {l, g, ln, gn, p}, n a
                natural number.
  -t T        Task over the logic:
                th --> prove theoremhood of the formula
                  in param. -f,
                c -->  prove consequence of the formula
                  in -f from the premises in param. -p,
```

```
                s -->  assignment of a model that
                  satisfies the set of equations in
                  param. -eq.
-f F        Formula in prefix notation.
            Constants as <a.b;c>, where a.b is the
               value
              (real) in the component and c is the
              component id starting from 0.
              T = true, F = false.
-p P        List of formulas in prefix notation
              -possibly with constants-.
-eq EQ      List of equations in the format
              (= formula formula') or
              (greater formula formula')
              -possibly with constants-.
-out OUT    Name for the output file.
```

We proceed to give the details of each one of these input arguments. If the reasoner is executed as

```
>python ctnormsolver.py [-l logic] [-t task] [-f formula]
    [-p premises] [-eq equations] [-out outputfile]
```

the details and meaning of each of these parameters is the following:

- **Logic**: The logic L the solver will work with can be specified in two ways.

  If the user desires to use BL logic, the input on this parameter should be fixed as `bl`. Internally, it will be interpreted as $(n+1)$`l` -i.e., the sum of $(n+1)$ Łukasiewicz components-, where n is the number of variables involved on the formulas/equations (see Theorem 2.23).

  Second, for determining a continuous t-norm based fuzzy logic, the input will consist on a word over the language $L \coloneqq \mathfrak{L} \setminus \{$`p1`$\}$ (see Definition 2.28), i.e. on a sequence of characters of the form `"[`c_0`+`c_1`+...+`c_n`]"` where each component $c_i \in \{$`l,g,l2,l3,...,g2,g3,...,p`$\}$ and `[`, `]` and + are literal symbols. The logic L will be determined by the logic of **Alg**(w).

  If the logic parameter is not specified, the default value is `"[l2]"`, i.e. the bi-valued classical logic.
- **Task**: The task the solver must perform over the input data can be chosen among three available:
  - **`th`**: theoremhood proving of the formula (in parameter `-f`) in the logic L (`-l`),
  - **`c`**: logical consequence of the formula (in parameter `-f`) from the premises (`-p`) in the logic L (`-l`),

  - **s**: satisfiability checking and model generation of the equations (in parameter `-eq`) in L (`-l`).

  If the task is not specified, the default value is `th`

- **Formula**: A formula should be specified whenever the task has been fixed to either `th` or `c`.

  The format of the formula shall be in polish notation with brackets, where the variables are of the form `x1,x2...` and the primitives are the following:
  - **tnorm**: `(tnorm x1 x1)` $\equiv (x1 * x1)$
  - **impl**: `(impl x1 x1)` $\equiv (x1 \to x1)$
  - **con**: `(con x1 x1)` $\equiv (x1 \wedge x1)$
  - **dis**: `(dis x1 x1)` $\equiv (x1 \vee x1)$
  - **neg**: `(neg x1)` $\equiv (\neg x1)$
  - **T**: `T` $\equiv \top$
  - **F**: `F` $\equiv \bot$

  Instead of a variable, a constant may be used, but the user will be responsible for its coherence. Constants will be specified on the form $< a.b; c >$, where $a.b$ is a value in the BL-chain specified by the component with index c in the ordinal sum that fixes the logic, and where $<, >$ and `;` are literal symbols. Notice that the values a, b are natural numbers (with b possibly equal to `0`), and c is a natural number. If the user desires to specify a constant with negative value, it shall be given in the form $< (\sim v); c >$, where $(,)$ and $\sim$ are literal symbols.

  The top element of the logic is represented by the character `T`, and the bottom element by `F`.

  If the formula is not specified, the default value is `T`.

- **Premises**: A list of formulas shall be given whenever task `c` (logical consequence) has been selected. The structure will be `"[`f_1`,` f_2`, ...,` f_n`]"`, where each f_i is a formula in the format detailed in the previous point and `[, ]` and `,` are literal symbols.

  If the list of premises is not specified, the default value will be the empty list (`"[]"`).

- **equations**: A list of equations shall be given whenever task `s` (satisfiability and model generation) has been selected. The structure will be `"[`eq_1`,` eq_2`, ...,` eq_n`]"`, where each eq_i is an equation and and `[, ]` and `,` are literal symbols. Each equation is of the form `(=` f_1 f_2`)` (where `(,)` and `=` are literal symbols), with two possibilities: either f_1, f_2 are formulas in the previous format or f_1= `(greater` q_1 q_2`)` with q_1, q_2 formulas in the previous format and f_2= `T`. This last format just represents the intuitive equation $q_1 > q_2$ in the algebra of the logic L.

  If the list of equations is not specified, the default value will be the empty list (`"[]"`).

- **outputfile**: The name for generating an output file, inside the `z3Codes` folder, with the SMT codes (in z3 input format) may be given.
  If the name of an output file is not specified, a file called `output` will be generated in the `z3Codes` folder for internal uses.

5.3.3. *Internal details.* Running the program with the desired options, an output file will be generated in the `z3codes` folder, either named `output` or with the string given in parameter `-out`.

The generated file will be different depending on all the attributes given, each of them affecting a certain section of code.

The first part of the file will be the same for all tasks and logics. In the case of working with the option to generate a model (`-t s`), an extra line

```
(set-option :produce-models true)
```

will be added in the head of the file to specify we want this option.

This first common section of code includes the theory selection, type and constants `T` and `F` definitions, and also the definitions of `min` and `max` operations (and `con` and `dis` too, since they are considered as basic operations to fasten the reasoner). We also define a function `greater` that represents the $>$ relation between pairs in the order of the BL-chain. It is the following:[2]

Listing 2. Common Code

```
(set-logic AUFLIRA)
;Pair type definitions
(define-sort Pair () (Array Int Real))
(define-fun p1 ((x Pair)) Real
 (select x 1))
(define-fun p2 ((x Pair)) Real
 (select x 2))
;truth constants
;T := <0,-1>
(declare-const T Pair)
 (assert (= T (store (store T 1 0.0) 2 (~ 1.0))))
;F = <0,0>
(declare-const F Pair)
 (assert (= F (store (store F 1 0.0) 2 0.0)))

;min(x,y)
(define-fun min ((x Pair) (y Pair)) Pair
 (ite (= x T) y
```

[2]For easier understanding of the following pieces of code, consider than `(ite c x y)` means `if c then x else y`, and ";" holds for commented lines on the code.

```
    (ite (= y T) x
     (ite (< (p2 x) (p2 y)) x
      (ite (< (p2 y) (p2 x)) y
       (ite (<= (p1 x) (p1 y)) x y))))))

;max(x,y)
(define-fun max ((x Pair) (y Pair)) Pair
 (ite (= (min x y) x) y x))

;conjunction min (x, y)
(define-fun con ((x Pair) (y Pair)) Pair
 (min x y))

;disjunction max (x, y)
(define-fun dis ((x Pair) (y Pair)) Pair
 (max x y))

;greater(x,y) T if x > y, F else
(define-fun greater ((x Pair) (y Pair)) Pair
 (ite (and (= x T) (> (p2 y) (~ 1.0))) T
  (ite (> (p2 x) (p2 y)) T
   (ite (and (= (p2 x) (p2 y)) (> (p1 x) (p1 y))) T
    F))))
```

The next section of the output file, defining the tnorm and implication operations and the variable limitations depending on each component, depends directly on the specified logic. In the case this one is BL, also slightly on the formulas involved. In this last case, the logic will be the ordinal sum of $(n+1)$ Łukasiewicz components, where n is the total number of different variables involved.

The t-norm operation is implemented as detailed in Definition 2.14. The variables are assigned as specified in the same definition, as pairs of the type $\langle value, componentID \rangle$ using an auxiliary integer variable for limiting the real values to the integer ones in the product components.

The following example illustrates the definition of the operations and variable limitations for the logic defined by the ordinal sum $\Pi \oplus G \oplus Ł_5$ (i.e, `-l "[p,g,l5]"`), working with formulas with `x1` as the only variable.

Listing 3. Operations and variables code for p+g+l5

```
;tnorm(x, y)
(define-fun tnorm ((x Pair) (y Pair)) Pair
 (ite (= (p2 x) (p2 y))
 ;x and y are in the same component
  (ite (= (p2 x) 0.0)
   (store x 1 (ite (<= (p1 x) (p1 y))
```

```
                (p1 x)
                (p1 y)))
    (ite (= (p2 x) 1.0)
     (store x 1 (+ (p1 x) (p1 y)))
     (ite (= (p2 x) 2.0)
    (store x 1 (ite (<= (p1 x) (p1 y))
                  (p1 x)
                  (p1 y)))
    (ite (= (p2 x) 3.0)
     (store x 1 (ite (>= (- (+ (p1 x) (p1 y)) 4.0) 0.0)
                 (- (+ (p1 x) (p1 y)) 4.0) 0.0))
                 x))))
   ;x and y are in different components
   (min x y)))

;impl(x y) -residuum-
(define-fun impl ((x Pair) (y Pair)) Pair
 (ite (or (= (min x y) x) (= x y))
  ;x<=y
  T
  ;y>x
  (ite (or (> (p2 x) (p2 y)) (= x T))
   ;x and y are in different components or x==T
   y
   ;x and y are in the same component
   (ite (= (p2 x) 0.0)
    y
    (ite (= (p2 x) 1.0)
     (store x 1 (- (p1 y) (p1 x)))
     (ite (= (p2 x) 2.0)
      y
      (ite (= (p2 x) 3.0)
       (store x 1 (+ 4.0 (- (p1 y) (p1 x))))
       x)))))))

;negation (neg x = x -> 0)
(define-fun neg ((x Pair)) Pair
 (impl x F))


;;;variables definition

(declare-fun x1 () Pair)
;generator for product-t values
(declare-fun x1i () Int)
```

```
(assert (< x1i 0))
(assert (or (= x1 T)
            (= (p2 x1) 0.0)
            (= (p2 x1) 1.0)
            (= (p2 x1) 2.0)
            (= (p2 x1) 3.0)))
(assert (ite (= (p2 x1) 0.0)
         (= (p1 x1) 0.0)
         (ite (= (p2 x1) 1.0)
          (= (p1 x1) (to_real x1i))
          (ite (= (p2 x1) 2.0)
           (and (< (p1 x1) 1.0) (>= (p1 x1) 0.0))
           (ite (= (p2 x1) 3.0)
            (or (= (p1 x1) 0.0) (= (p1 x1) 1.0) (= (p1 x1
               ) 2.0) (= (p1 x1) 3.0) )
            (= x1i x1i))))))
```

If any constant has been used in the specification of the inputs (either in the formula, the premises or the equations), integrated on the previous code, an extra fragment will be generated. It will consist on the translation of the input the user has given to a code understandable by the z3-solver, assigning an unique name `ci` (for `i` a natural number) to the pair of values. Also, the apparition of each constant in the formula or equation will be substituted by the new constant name. This code is of the same kind that the one used for the definition of the `T` and `F` constants.

The following example illustrates the code generated by two constants in the formulas involved, `<0.4;0>` and `<0.26;0>`.

Listing 4. Constants <0.4;0> and <0.26;0>

```
(declare-const c0 Pair)
(declare-const c1 Pair)

...
(assert (= c0 (store (store c0 1 0.4) 2 0.0)) )
(assert (= c1 (store (store c1 1 0.26) 2 0.0)) )
```

After the definitions of the operations, variables and constants, the skeleton of the generated file is different depending directly on the option specified.

If the option leads to work with theoremhood or with logical consequence (`-t th` or `-t c`), they have a common affirmation, namely

Listing 5. Theoremhood

```
(assert (= (greater T <form>) T))
```

where ⟨form⟩ is the formula specified in attribute `-f`.

With this, we are asserting that the value of the formula is not always equal to `T` (literally we are affirming that $T >$ ⟨form⟩). If this condition is unsatisfiable, it means that all evaluations of the formula are equal to `T`.

If we are in the case of proving this formula is the logical consequence of a set of premises, the generated code will also include the sentences

Listing 6. Logical consequence

```
(assert (= <prem1> T))
(assert (= <prem2> T))
...
```

that limit the models where the formula will be verified to the ones that make all the premises true.

In both cases, after this, the order

```
(chek-sat)
```

is given to the solver so it starts working with all the given information to reach a solution.

On the other hand, if the operation to do is checking the satisfiability and generating a model of a set of equations, the code will consist in the assertion of the given equations as specified by the user and the order to start reasoning with them, and afterwards, a final section of code to get the values of the model.

Listing 7. Model generation

```
(assert eq1)
(assert eq2)
...
(check-sat)

(get-value ((p2 x1)))
(get-value ((p1 x1)))
...
```

5.3.4. *Output.* Depending on the task we are performing, the output of the program will be different.

- If the task was specified for proving theoremhood (`-t th`), the output message will be either the confirmation or the negation that the specified formula is a theorem in the given logic. Namely,

```
Formula <formula> IS a theorem in logic <logic>
Formula <formula> IS NOT a theorem in logic <logic
    >
```

- Similarly, if the task to be done was proving logical consequence (`-t c`), the output message will be either the confirmation or the negation that the specified formula is a logical consequence of the premises in the given logic. Namely,

```
Formula <formula> IS a logical consequence of
  <premises> in logic <logic>
Formula <formula> IS NOT a logical consequence of
  <premises> in logic <logic>
```

- If the task was specified for checking satisfiability and generating a model of a set of equations (`-t s`), the output message will either confirm or deny that the equations set is satisfiable, and in case it is, it will show the values of the variables in some generated model that satisfies them. These values will have the same format of the constants that could be specified in the input, i.e. $< v;c >$ where v is the value in the algebra corresponding to the component of the ordinal sum (parameter `-l`) of index c -starting in 0-. Notice that, in the case of BL(`-l bl`), this answer depends on the codification of BL as $(n+1)$`l`, and in the case any component of Product logic is involved in the word for specifying the logic, all the components will suffer its index translation depending on the codification of the product components as two items (since `p` components are codified as `l2+Z-`).

  The output will be shown as:

```
The set of equations <equations> DOES have a model
    in logic <logic> and it is:
x1 --> <v1;c1>
x2 --> <v2;c2>
...
The set of equations <equations> DOES NOT have a
   model in logic <logic>
```

In the case the evaluation of a variable is either $\top$ or $\bot$, the output will directly show this with `T` or `F` respectively, i.e.

```
x1 --> T
x2 --> F
```

5.3.5. *Examples.* For an easier comprehension on the usage of the solver, several examples will be provided with its logical meaning and its corresponding output.

- $\models_{\mathrm{BL}} (x_1 * x_2) \to (x_2 * x_1)$

```
>python ctnormsolver.py -l bl -t th -f "(impl (
   tnorm x1 x2) (tnorm x2 x1))"
```

---

will provide the output

```
Formula (impl (tnorm x1 x2) (tnorm x2 x1)) IS a
   theorem in logic bl
```

- $\models_{\mathrm{BL}} \neg\neg x_1 \to x_1$

```
>python ctnormsolver.py -l bl -t th -f "(impl (neg
     (neg x1)) x1)"
```

---

will provide the output

```
Formula (impl (neg (neg x1)) x1) IS NOT a theorem
   in logic bl
```

- $\models_{[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}} \neg\neg x_1 \to x_1$

```
>python ctnormsolver.py -l [l] -t th -f "(impl (
   neg (neg x1)) x1)"
```

---

will provide the output

```
Formula (impl (neg (neg x1)) x1) IS a theorem in
   logic [l]
```

- $\models_{[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}} \oplus [\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}} \oplus \mathbf{G_5} \oplus [\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}} (x_1 \to x_2) \to ((x_2 \to x_3) \to (x_1 \to x_3))$

```
>python ctnormsolver.py -l [l+p+g5+p] -t th -f "(
   impl (impl x1 x2) (impl (impl x2 x3) (impl x1
   x3)))"
```

---

will provide the output

```
Formula (impl (impl x1 x2) (impl (impl x2 x3) (
   impl x1 x3))) IS a theorem in logic [l+p+g5+p]
```

- $x_1, x_2 \to x_3 \models_{[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}} \oplus [\mathbf{0},\mathbf{1}]_{\mathbf{\Pi}}} (x_1 \to x_2) \to x_3$

```
>python ctnormsolver.py -l [l+p] -t c -p "[x1,(
   impl x2 x3)]" -f "(impl (impl x1 x2) x3)"
```

---

will provide the output

```
Formula (impl (impl x1 x2) x3) IS a logical
   consequence of [x1,(impl x2 x3)] in logic [l+p]
```

- $\langle 0.4, 0\rangle \to x_1, x_1 \to x_2 \models_{[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}} \langle 0.26, 0\rangle \to x_2$

```
>python ctnormsolver.py -l [l] -t c -p "[(impl
   <0.4;0> x1), (impl x1 x2)]" -f "(impl
   <0.26;0> x2)"
```

will provide the output

```
Formula (impl <0.26;0> x2) IS a logical
   consequence of [(impl <0.4;0> x1), (impl x1 x2)
   ] in logic [l]
```

- $\textbf{model}_{[\mathbf{0},\mathbf{1}]_{\mathbf{L}}}\big((x_1 \wedge x_2) \to (x_1 * x_2), (x_1 > \ F), (T > x_1)\big)$

```
>python ctnormsolver.py -l "[l]" -t s -eq "[(=(
   impl (con x1 x2) (tnorm x1 x2)) T),(= (greater
   x1 F) T), (= (greater T x1) T)]"
```

will provide the output

```
The set of equations [(=(impl (con x1 x2) (tnorm
   x1 x2)) T),(= (greater x1 F) T),
(= (greater T x1) T)] DOES have a model in logic [
   l]
A possible assignation in the BL-chain assignated
   to
[l] of the involved variables is the following:
x2 --> T
x1 --> <0.5;0.0>
```

5.4. **Experimental Results**

One of the strongest points of this solver is the versatility it has, allowing, as commented, theoremhood proofs, logical deduction proofs and model generation tests. Several tests on all the options were run for checking its proper codification, but the intensive testing performed for efficiency studies performed is done using only its theoremhood prover option, to compare its results to [ABMV12].

Since all possible theorems on BL so are on any of its extensions, experiments over two different families of BL-theorems were conducted, see (2) and (3) below. First, for comparison reasons with [ABMV12], the following generalizations (based on powers of the & connective) of the first seven Hájek's axioms of BL [Háj98] were considered:

$$
\begin{array}{lll}
 & (\mathrm{A1}) & (p^n \to q^n) \to ((q^n \to r^n) \to (p^n \to r^n)) \\
 & (\mathrm{A2}) & (p^n \& q^n) \to p^n \\
 & (\mathrm{A3}) & (p^n \& q^n) \to (q^n \& p^n) \\
(2) & (\mathrm{A4}) & (p^n \& (p^n \to q^n)) \to (q^n \& (q^n \to p^n)) \\
 & (\mathrm{A5}a) & (p^n \to (q^n \to r^n)) \to ((p^n \& q^n) \to r^n) \\
 & (\mathrm{A5}b) & ((p^n \& q^n) \to r^n) \to (p^n \to (q^n \to r^n)) \\
 & (\mathrm{A6}) & ((p^n \to q^n) \to r^n) \to (((q^n \to p^n) \to r^n) \to r^n)
\end{array}
$$

where p, q and r are propositional variables, and $n \in \mathbb{N} \setminus \{0\}$. It is worth noticing that the length of these formulas grows linearly with the parameter n.

In [ABMV12] the authors refer to [Rot07] to justify why these formulas can be considered a good test bench for (at least) Łukasiewicz logic. In our opinion, these formulas have the problem to be a good evaluator set of using only three variables. We consider this is a serious drawback because the known results on BL complexity (see Chapter 3) state that Łukasiewicz-SAT is an NP-complete problem when the number of variables in the input is not fixed. However, we consider that proving that tautologicity for formulas with three variables can be solved in polynomial time could be done.

With this in mind, to overcome the drawback of the bounded number of variables, a new family of BL-theorems to be used as a bench test too is presented.

For every $n \in \mathbb{N} \setminus \{0\}$,

$$
\bigwedge_{i=1}^{n} (\&_{j=1}^{n}\, p_{ij}) \;\to\; \bigvee_{j=1}^{n} (\&_{i=1}^{n}\, p_{ij}) \tag{3}
$$

is a BL-theorem which uses n^2 variables; the length of these formulas grows quadratically with n. As an example, we note that for $n = 2$ we get the BL-theorem $\big((p_{11}\&p_{12}) \wedge (p_{21}\&p_{22})\big) \to \big((p_{11}\&p_{21}) \vee (p_{12}\&p_{22})\big)$. These formulas can be considered significantly harder than the ones previously proposed in [Rot07]; and indeed, the experimental results support this claim. It is important to notice that the natural way to compare this new formula with parameter n with the previous set is to consider the formulas in [Rot07] with the integer part of $\sqrt{n}$ as parameter.

5.4.1. *Data.* Experiments were run on a machine with a i5-650 3.20GHz processor and 8GB of RAM. Evaluating the validity in Łukasiewiczand Gödel logics of the generalizations of the BL axioms (2), ranging n from 0 to 500 with increments of 10, throws better results than the ones obtained in [ABMV12], but since the new solver is, on these logics, an extension of their work, this can be assumed to be due to the use of different machines.
For Product Logic, really good timings were obtained. Actually, they are

worse than the ones for Łukasiewicz and Gödel logics in most of the cases, since the Presburger arithmetic has high complexity too, but the difference with the previous approach is clear: complex formulas are solved in a comparatively short time, whereas in [ABMV12] they could not even be processed. In Figure 1 one can see and compare solving times (given in seconds) for some of the axioms of the test bench for the cases of BL, Łukasiewicz, Gödel and Product logics. It is also interesting to observe how irregularly the computation time for Product Logic varies depending on the axiom and the parameter. This probably happens due to the way the z3-solver internally works with the integer arithmetic theory.

The experiments done with the other family of BL-theorems (3) (see Figure 2 for the results) suggests that here the evaluation time is growing non-polynomially on the parameter n. In the graphs we give here, only those answers (for parameters $n \leq 70$) obtained in at most 3 hours of execution are shown (e.g. for the BL case answers could be reached within this time only for the problems with $n \leq 4$). The high differences in time when evaluating the theorems were expectable: Łukasiewicz and Gödel are simpler than BL when proving the theoremhood because of the method used for BL (considering $n^2 + 1$ copies of Łukasiewicz, where n is the parameter of the formula). On the other hand, the computation times for Product logic modelled over $\mathcal{Z}^-$ are also smaller than for BL.

5.5. **The discrete solver**

We want to notice here a failed attempt on enhancing this results. Seeing the good results obtained when working with a discrete theory instead of a continuous one (in the case of the product logic), we decided to generalize this approach to the other continuous components.

We implemented and tested a "discretized" version of this solver following the results presented on Theorems 2.24,2.25 and 2.26. In this version not only the Product components but also the Łukasiewicz and the Gödel ones ($[\mathbf{0},\mathbf{1}]_{\mathbf{Ł}}$ and $[\mathbf{0},\mathbf{1}]_{\mathbf{G}}$, valued over the real interval $[0,1]$) were computed over a discrete set of values, using the theory of the natural numbers in the SMT-solver.

However,the results were far from being as good as the ones obtained with the continuous implementation. The problem with this approach was that the bound for using the finite logics in concrete problems is too high (it grows exponentially on the number of variables), and so, the results were sadly much worse than the ones obtained with the mixed countable/uncountable(Łand G, Π Ł$_n$ G_n) approach, which is the one that was finally selected as the best one. In [SJVC09] the worst-case upper bound of Aguzzoli et al. for the formulation of Theorem 2.24 is improved, but in our case, this approach continues being unuseful. In most of the cases, the test cases presented in Section 5.4 were impossible to be processed by the z3 solver, and for this

reason we keep this version as an interesting but not successful trial for solving the problem.

We comment here this failed application to avoid future approaches on this matter, given that at least for this kind of problems, this work is not useful from an efficiency point of view.

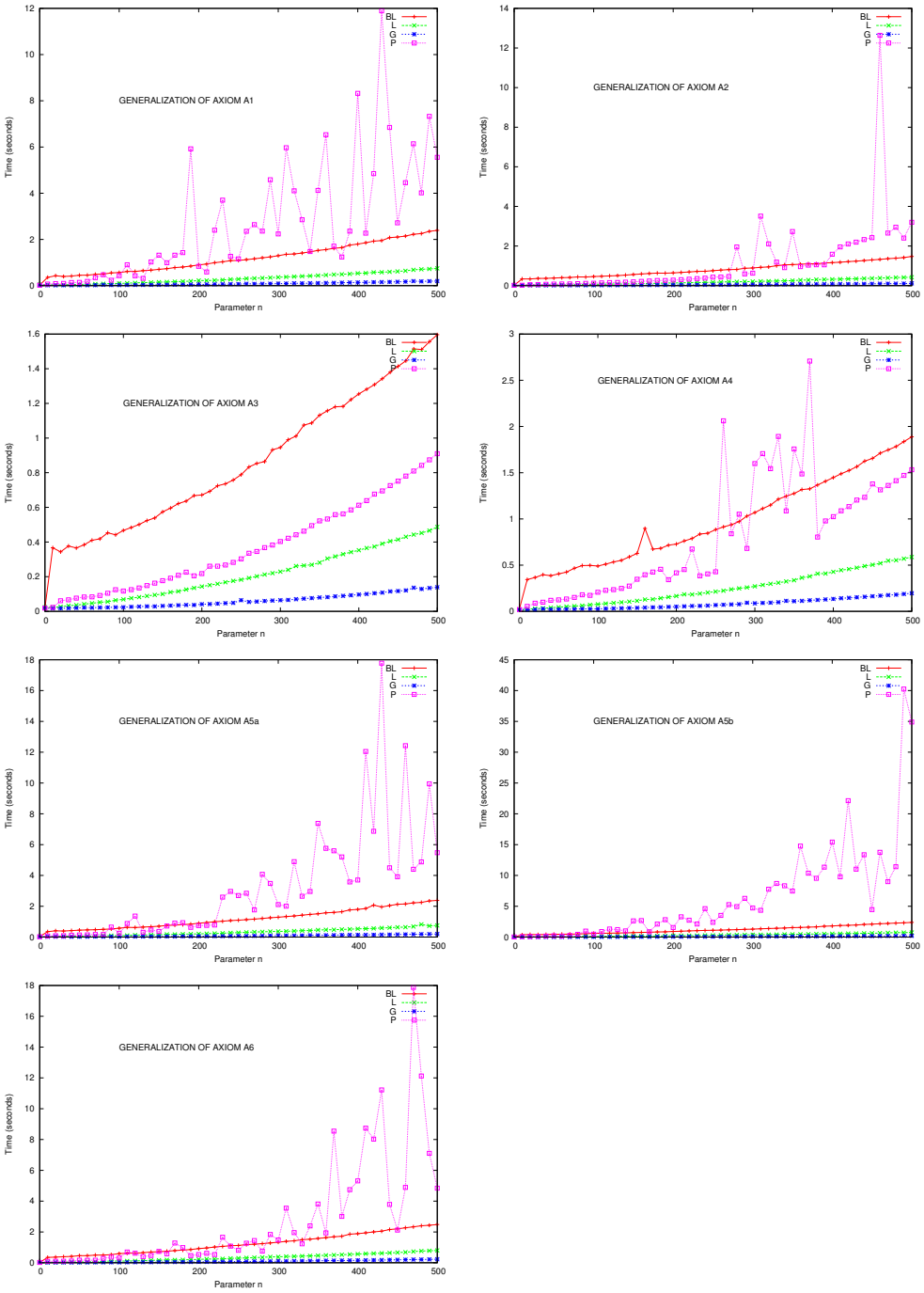

FIGURE 1. Generalizations of BL-axioms given in (2).

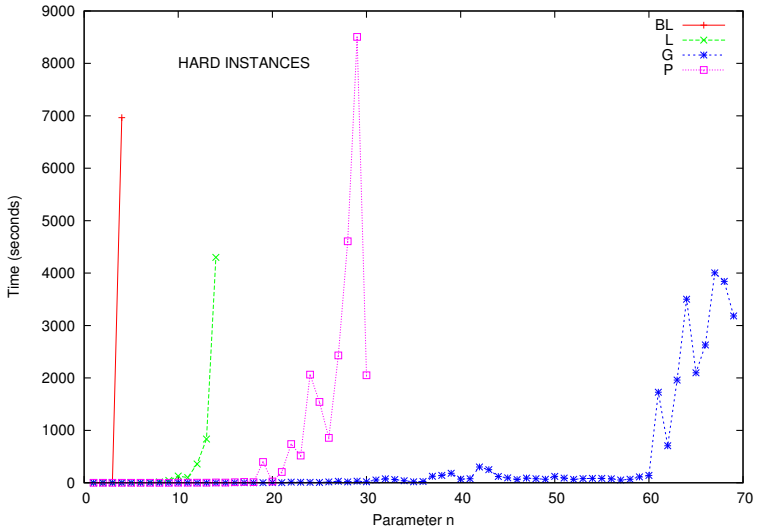

FIGURE 2. Our proposed BL-theorems given in (3).

6. Related Works and Conclusions

We provide here an overview of existing works that cope with similar problems that the one presented in Chapter 5, to be able to comparatively study the possible impact of this research.

Also we detail the conclusions reached after this research, and provide several points for future work that would complete this study.

6.1. **Related Works**

To the best of our knowledge, little effort has been paid to develop a working system for reasoning about formal fuzzy logics.

Theoretical results have been presented to reason about fuzzy logics, providing interesting calculus methods and approaches [Rot07, SJVC09, BS09, SJV12] but there are no available implementations and working applications over these results.

On the other hand, there is an interesting tool for working with fuzzy ontologies, related but not equivalent to a formal fuzzy logic reasoner [DeL]. No more works on this field could be found, and so it is reasonable to consider this work on a continuous t-norm based fuzzy logics reasoner is a very interesting contribution for the community.

In works from Bobillo, Straccia et al. [BS11, BDGR12] implementations for reasoning with fuzzy ontologies have been presented, specifically, with fuzzy rough Description Logics (DL).

Description logics [BCM+07] are a family of logics for representing structured knowledge. Each logic is denoted by using a string of capital letters which identify the constructors of the logic and therefore its expressiveness. DLs have proved to be very useful as ontology languages.

In computer science and information science, an ontology formally represents knowledge as a set of concepts within a domain, and the relationships between those concepts. It can be used to reason about the entities within that domain and may be used to describe the domain. Ontologies are the structural frameworks for organizing information and are used in Artificial Intelligence, the Semantic Web, system and software engineering and other fields referring to information treatment as a form of ad-hoc knowledge representation about the world or some part of it.

A presented reasoner, DeLorean ([DeL]) is a DL reasoner that supports fuzzy rough extensions of the existing fuzzy DLs. In a strict sense, DeLorean is not a reasoner but a translator from a fuzzy rough ontology language into a classical ontology language (OWL or OWL 2). Then, a classical DL reasoner is employed to reason with the resulting ontology.

This approach to fuzzy logic reasoners provides an intuitive approach to the modelling of vague problems, but on the other hand it is not clear how

to work with t-norm based fuzzy logics formally defined (as for example BL or over BL logics).

Also, as it is shown in the specifications of DeLorean, it does not work with infinitely-valued logics, since in its language, a finite chain of truth degrees is assumed. This is a serious problem, making impossible to work within, for example, Product Logic. Besides, as we will comment in Section 5.2, although the reasoning for concrete instances in Łukasiewicz and Gödel infinitely valued logics is possible in finite algebras, it is in general not efficient as it will be seen in the experiments shown in Section 5.4.

For these reasons, this research is oriented to neighbouring but very different problems, and so the solver presented in Chapter 5 should be considered an alternative reasoner system.

6.2. **Conclusions**

From a general point of view, concerning the impact of this research, we found remarkable the fact that there is no other existing system which scope of problems coincide with the ones treated by the solver presented as main point of this work. This leads this result to be opening an interesting field for both the AI (or more generally, the computer science) and the logical communities, namely the concrete treatment of problems from a wide family of logics in a purely automatized way. To remark the previous statement, it is natural to think that the range of users of the new solver is more theoretical-oriented than the one from reasoners like DeLorean, just because of the logical background that is necessary to know and to be interested in studying the family of logics this new application works with.

On the other hand, we reached several intermediate conclusions during the research, that can be briefly[3] exposed here.

(1) Logics from the BL-chains obtained by the classical t-norms ordinal sum can be equally expressed as hoops ordinal sums, allowing richer decomposition of the components.

(2) From the z3 SMT-solver point of view (and possible in general), to work with a concrete constraint satisfaction problem in an uncountable set with product and division operations is much slower than treating an equivalent problem (proved of the same computational complexity) in a discrete universe with the addition and subtraction.

(3) Reductions to equivalent problems to work within discrete or finite universes is not always the best option, since the hardnesses of the original problem can be translated to even more problematic points.

6.3. **Future Work**

Some interesting features can be very easily added to the reasoner as it is implemented by now. The theoretical work that should be done to proof

[3]For deeper details, see its correspondent section

they would be correct (in case that is necessary) does not seem like it could be a problem (cf. [Mon01], [EGGN07])

The ones we consider more relevant are the following:

(1) Implementation a graphical interface for the solver to make the application more accessible and user-friendly.
(2) Extension on the solver for allowing the specification of the separation points for the basic t-norms in the input, and the use of constants in $[0, 1] \cap \mathbb{Q}$ for words that do not involve the product component.
(3) Extension of the solver to natively include the Baaz Projector Δ.
(4) Extension of the solver for allowing the specification of infinite number of components in the word definition. The theoretical work behind this point is already done, and just the parsing for the computer application inputs should be implemented.

Another idea we consider interesting to study is the extension or adaptation of this research to the development of a fuzzy modal logic solver. This kind of logics have been studied for the modelization of concepts such as beliefs for agents ([Cas08], [GHE03]) and could be very interesting for the implementation of this theoretical works.

References

[ABMV12] C. Ansótegui, M. Bofill, F. Manyà, and M. Villaret. Building automated theorem provers for infinitely valued logics with satisfiability modulo theory solvers. In *Proceedings of the IEEE 42nd International Symposium on Multiple-Valued Logic (ISMVL 2012)*. IEEE Computer Society, 2012.

[AC00] S. Aguzzoli and A. Ciabattoni. Finiteness in infinite-valued Łukasiewicz logic. *Journal of Logic, Language and Information*, 9(1):5–29, 2000. Logics of uncertainty.

[ACGM05] A. Armando, C. Castellini, E. Giunchiglia, and M. Maratea. A SAT-based decision procedure for the boolean combination of difference constraints. In *Proceedings of the 7th international conference on Theory and Applications of Satisfiability Testing*, SAT'04, pages 16–29. Springer-Verlag, 2005.

[AFM07] P. Aglianò, I. M. A. Ferreirim, and F. Montagna. Basic hoops: an algebraic study of continuous t-norms. *Studia Logica*, 87(1):73–98, 2007.

[AG02] S. Aguzzoli and B. Gerla. On countermodels in Basic Logic. *Neural Network World*, 12(5):407–421, 2002.

[AGH05] S. Aguzzoli, B. Gerla, and Z. Haniková. Complexity issues in basic logic. *Soft Computing*, 9(12):919–934, 2005.

[Agu04] S. Aguzzoli. Uniform description of calculi for all t-norm logics. In *ISMVL '04*, pages 38–43, 2004.

[AM03] P. Agliano and F. Montagna. Varieties of BL-algebras. I. General properties. *Journal of Pure and Applied Algebra*, 181(2-3):105–129, 2003.

[AS09] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 399–404. Morgan Kaufmann Publishers Inc., 2009.

[BCLZ04] T. Ball, B. Cook, S. Lahiri, and L. Zhang. Zapato: Automatic theorem proving for predicate abstraction refinement. In *16th International Conference on Computer Aided Verification (CAV 2004)*, volume 3114 of *Lecture Notes in Computer Science*, pages 457–461. Springer, 2004.

[BCM+07] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, second edition, 2007.

[BDGR12] F. Bobillo, M. Delgado, and J. Gómez-Romero. Delorean: A reasoner for fuzzy owl 2. *Expert Systems with Applications*, 39(1):258 – 272, 2012.

[BEGR11] F. Bou, F. Esteva, L. Godo, and R. Rodríguez. On the minimum many-valued modal logic over a finite residuated lattice. *Journal of Logic and Computation*, 21(5):739–790, 2011.

[BHMV02] M. Baaz, P. Hájek, F. Montagna, and H. Veith. Complexity of t-tautologies. *Annals of Pure and Applied Logic*, 113(1-3):3–11, 2002. First St. Petersburg Conference on Days of Logic and Computability (1999).

[BM79] R. S. Boyer and J. Strother Moore. *A computational logic*. Academic Press [Harcourt Brace Jovanovich Publishers], New York, 1979. ACM Monograph Series.

[BS00] S. Burris and H. P. Sankappanavar. *A course in Universal Algebra*. The millennium edition, 2000.

[BS09] F. Bobillo and U. Straccia. Fuzzy description logics with general t-norms and datatypes. *Fuzzy Sets and Systems*, 160(23):3382–3402, 2009.

[BS11] F. Bobillo and U. Straccia. Fuzzy ontology representation using owl 2. *International Journal of Approximate Reasoning*, 52(7):1073 – 1094, 2011.

[BST10] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `www.SMT-LIB.org`.

[Cas08] A. Casali. *On Intentional and Social Agents with Graded Attitudes*. Ph. D. Dissertation, University of Girona (Spain), 2008.

[CEGT00] R. Cignoli, F. Esteva, L. Godo, and A. Torrens. Basic fuzzy logic is the logic of continuous t-norms and their residua. *Soft Computing*, 4(2):106–112, 2000.

[CH09] P. Cintula and P. Hájek. Complexity issues in axiomatic extensions of Łukasiewicz logic. *Journal of Logic and Computation*, 19(2):245–260, 2009.

[CT00] R. Cignoli and A. Torrens. An algebraic analysis of product logic. *Multiple-valued logic*, 5:45–65, 2000.

[CT05] R. Cignoli and A. Torrens. Standard completeness of Hájek basic logic and decompositions of BL-chains. *Soft Computing*, 9(12):862–868, 2005.

[DdM06] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *CAV*, pages 81–94. Springer, 2006.

[DeL] DeLorean. http://webdiis.unizar.es/ fbobillo/delorean.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5:394–397, 1962.

[dMB08] L. Mendonça de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[dMDS07] L. de Moura, B. Dutertre, and N. Shankar. A tutorial on satisfiability modulo theories. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2007.

[dMR02] L. de Moura and H. Ruess. Lemmas on demand for satisfiability solvers. In *Proceedings of the 5th international conference on Theory and Applications of Satisfiability Testing*, SAT'02. Springer-Verlag, 2002.

[dMR04] L. de Moura and H. Ruess. An experimental evaluation of ground decision procedures. In R. Alur and D. Peled, editors, *Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts)*, volume 3114 of *Lecture Notes in Computer Science*, pages 162–174. Springer, 2004.

[DP60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960.

[EGGN07] F. Esteva, J. Gispert, L. Godo, and C. Noguera. Adding truth-constants to logics of continuous t-norms: Axiomatization and completeness results. *Fuzzy Sets and Systems*, 158(6):597–618, March 2007.

[ES03] N. Eén and N. Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[Fer92] I. Ferreirim. *On varieties and quasivarieties of hoops and their reducts*. Ph. D. Thesis, University of Illinois at Chicago, 1992.

[GHE03] L. Godo, P. Hájek, and F. Esteva. A fuzzy modal logic for belief functions. *Fundamenta Informaticae*, 57(2-4):127–146, 2003. 1st International Workshop on Knowledge Representation and Approximate Reasoning (KR&AR) (Olsztyn, 2003).

[GN07] E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat-solver. *Discrete Appl. Math.*, 155(12):1549–1561, 2007.

[Háj98] P. Hájek. *Metamathematics of fuzzy logic*, volume 4 of *Trends in Logic—Studia Logica Library*. Kluwer Academic Publishers, Dordrecht, 1998.

[McC62] J. McCarthy. Towards a mathematical science of computation. In *In IFIP Congress*, pages 21–28. North Holland, 1962.

[Mon01] F. Montagna. Free BL_Δ algebras. In A. Di Nola and G. Gerla, editors, *Lectures on soft computing and fuzzy logic*, Adv. Soft Comput., pages 159–171. Physica, Heidelberg, 2001.

[Mon05] F. Montagna. Generating the variety of BL-algebras. *Soft Computing*, 9(12):869–874, 2005.

[MS57] P. S. Mostert and A. L. Shields. On the structure of semigroups on a compact manifold with boundary. *Annals of Mathematics. Second Series*, 65:117–143, 1957.

[Mun87] D. Mundici. Satisfiability in many-valued sentential logic is NP-complete. *Theoretical Computer Science*, 52(1-2):145–153, 1987.

[NEGM05] A. Di Nola, F. Esteva, L. Godo, and F. Montagna. Varieties of BL-algebras. *Soft Computing*, 9(12):875–888, 2005.

[NO80] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *J. ACM*, 27(2):356–364, 1980.

[NOT06] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the Association for Computing Machinery*, 53:937–977, 2006.

[Pav79a] J. Pavelka. On fuzzy logic. I. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25(1):45–52, 1979. Many-valued rules of inference.

[Pav79b] J. Pavelka. On fuzzy logic. II. Enriched residuated lattices and semantics of propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25(2):119–134, 1979.

[Pav79c] J. Pavelka. On fuzzy logic. III. Semantical completeness of some many-valued propositional calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 25(5):447–464, 1979.

[Rot07] R. Rothenberg. A class of theorems in Łukasiewicz logic for benchmarking automated theorem provers. In N. Olivetti and C. Schwind, editors, *TABLEAUX '07, Automated Reasoning with Analytic Tableaux and Related Methods, Position Papers*, number LSIS.RR.2007.002, pages 101–111, 2007.

[SC] SMT-COMP. http://smtcomp.sourceforge.net/.

[Sho81] R. Shostak. Deciding linear inequalities by computing loop residues. *J. ACM*, 28(4):769–779, October 1981.

[SJV12] S. Schockaert, J. Janssen, and D. Vermeir. Satisfiability checking in Łukasiewicz logic as finite constraint satisfaction. *Journal of Automated Reasoning*, 49(4):493–550, 2012.

[SJVC09] S. Schockaert, J. Janssen, D. Vermeir, and M. De Cock. Finite satisfiability in infinite-valued Łukasiewicz logic. In L. Godo and A. Pugliese, editors, *SUM*, volume 5785 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2009.

[SSB02] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pages 209–222, London, UK, UK, 2002. Springer-Verlag.

[VBG12] A. Vidal, F. Bou, and L. Godo. An SMT-based solver for continuous t-norm based logics. In Springer, editor, *6th International Conference on Scalable Uncertainty Management (SUM)*, Lecture Notes in Computer Science, pages 633–640, 2012.

[z3w] Z3. http://research.microsoft.com/en-us/um/redmond/projects/z3/index.html.