An SMT-based solver for continuous t-norm based logics (extended version)

Amanda Vidal[1], Félix Bou[21], and Lluís Godo[1]

[1] Artificial Intelligence Research Institute (IIIA - CSIC)
Campus de la Universitat Autònoma de Barcelona s/n,
08193 Bellaterra, Spain
[2] Department of Probability, Logic and Statistics
Faculty of Mathematics, University of Barcelona (UB)
Gran Via 585, 08007 Barcelona, Spain

Abstract. In the literature, little attention has been paid to the development of solvers for systems of mathematical fuzzy logic, even though there is an important number of studies on complexity and proof theory for them. In this paper, extending a recent approach by Ansótegui et al., we present ongoing work on an efficient and modular SMT-based solver for a wide family of continuous t-norm based fuzzy logics. The solver is able to deal with most famous fuzzy logics (including BL, Łukasiewicz, Gödel and Product); and for each of them, it is able to test, among others, satisfiability, tautologicity and logical consequence problems. Note that, unlike the classical case, these problems are not in general interdefinable in fuzzy logics. Some empirical results are reported at the end of the paper.

1 Introduction

In the literature, with a few exceptions mainly for Łukasiewicz logics [13,15,4,14], little attention has been paid to the development of efficient solvers for systems of mathematical fuzzy logic, even though there is an important number of studies on complexity and proof theory for them (see [12,9,10,1,6]). This is a problem that limits the use of fuzzy logics in real applications. In [2], a new approach for implementing a theorem prover for Łukasiewicz, Gödel and Product fuzzy logics using Satisfiability Modulo Theories (SMT) has been proposed. The main advantage of this approach based on SMT is the modularity of being able to cope with several fuzzy logics.

In this paper, we extend this approach in order to be able to cope with more logics (including Basic Fuzzy Logic BL): we study the implementation and testing of a general solver for continuous t-norm based fuzzy logics. We have generalized the solver so it can perform satisfiability, theoremhood and logical consequence checks for any of a wide family of these fuzzy logics. Also, we have changed the coding for product logic from the one of [2] to one based on Presburger Arithmetic (Linear Integer Arithmetic), and this has dramatically enhanced its performance.

Structure of the paper. Section 2 introduces the propositional logics considered, and gives a brief introduction to SMT. Section 3 describes the SMT-based solver proposed in this paper. Section 4 starts with an explanation of the design of the experiments we ran on our solver, and then the results are analyzed in Section 4.1. Section 5 presents the conclusions and future work. Finally, in Appendix A we show some code in the SMT-LIB format: we point out that this code is missing in the proceedings version [17] of the present paper.

2 Preliminaries

2.1 Continuous t-norm based fuzzy logics

Continuous t-norm based propositional logics correspond to a family of many-valued logical calculi with the real unit interval $[0,1]$ as set of truth-values and defined by a conjunction &, an implication $\rightarrow$ and the truth-constant $\overline{0}$, interpreted respectively by a continuous t-norm $*$, its residuum $\Rightarrow$ and the number 0. In this framework, each continuous t-norm $*$ uniquely determines a semantical propositional calculus L_* over formulas defined in the usual way (see [10]) from a countable set $\{p, q, r, \ldots\}$ of propositional variables, connectives & and $\rightarrow$ and truth-constant $\overline{0}$. Further connectives are defined as follows:

$$\begin{aligned} \neg\varphi &\text{ is } \varphi \rightarrow \overline{0}, \\ \varphi \wedge \psi &\text{ is } \varphi \& (\varphi \rightarrow \psi), \\ \varphi \vee \psi &\text{ is } ((\varphi \rightarrow \psi) \rightarrow \psi) \wedge ((\psi \rightarrow \varphi) \rightarrow \varphi), \\ \varphi \equiv \psi &\text{ is } (\varphi \rightarrow \psi) \& (\psi \rightarrow \varphi). \end{aligned}$$

L_*-evaluations of propositional variables are mappings e assigning to each propositional variable p a truth-value $e(p) \in [0,1]$, which extend univocally to compound formulas as follows: $e(\overline{0}) = 0$, $e(\varphi \& \psi) = e(\varphi) * e(\psi)$ and $e(\varphi \rightarrow \psi) = e(\varphi) \Rightarrow e(\psi)$. Actually, each continuous t-norm defines an algebra $[0,1]_* = ([0,1], \min, \max, *, \Rightarrow, 0)$, called *standard* L_*-algebra.

From the above definitions it holds that $e(\neg\varphi) = e(\varphi) \Rightarrow 0$, $e(\varphi \wedge \psi) = \min(e(\varphi), e(\psi))$, $e(\varphi \vee \psi) = \max(e(\varphi), e(\psi))$ and $e(\varphi \equiv \psi) = e(\varphi \rightarrow \psi) * e(\psi \rightarrow \varphi)$. A formula φ is a said to be a *1-tautology* (or *theorem*) of L_* if $e(\varphi) = 1$ for each L_*-evaluation e. The set of all 1-tautologies of L_* will be denoted as $TAUT([0,1]_*)$. A formula φ is *1-satisfiable* in L_* if $e(\varphi) = 1$ for some L_*-evaluation e. Moreover, the corresponding notion of *logical consequence* is defined as usual: $T \models_* \varphi$ iff for every evaluation e such that $e(\psi) = 1$ for all $\psi \in T$, $e(\varphi) = 1$.

Well-known axiomatic systems, like Łukasiewicz logic (Ł), Gödel logic (G), Product logic (Π) and Basic Fuzzy logic (BL) syntactically capture different sets $TAUT([0,1]_*)$ for different choices of the t-norm $*$ (see e.g. [10,6]). Indeed, the following conditions hold true, where $*_{\text{Ł}}$, $*_G$ and $*_\Pi$ respectively denote the Łukasiewicz t-norm, the min t-norm and the product t-norm:

φ is provable in Ł iff $\varphi \in TAUT([0,1]_{*_{Ł}})$
φ is provable in G iff $\varphi \in TAUT([0,1]_{*_G})$
φ is provable in Π iff $\varphi \in TAUT([0,1]_{*_\Pi})$
φ is provable in BL iff $\varphi \in TAUT([0,1]_{*})$ for all continuous t-norms $*$.

Also, taking into account that every continuous t-norm $*$ can be represented as an ordinal sum of Łukasiewicz, Gödel and Product components, the calculus of any continuous t-norm has been axiomatized in [8]. All these completeness results also hold from deductions from a finite set of premises but, in general, they do not extend to deductions from infinite sets (see [6] for details).

2.2 Satisfiability Modulo Theories (SMT)

The satisfiability problem, i.e. determining whether a formula expressing a constraint has a solution, is one of the main problems in theoretical computer science. If this constraint refers to Boolean variables, then we are facing a well-known problem, the (propositional) Boolean satisfiability problem (SAT).

On the other hand, some problems require to be described in more expressive logical languages (like those of the first order theories of the real numbers, of the integers, etc.), and thus a formalism extending SAT, Satisfiability Modulo Theories (SMT), has also been developed to deal with these more general decision problems. An SMT instance is a first order formula where some function and predicate symbols have predefined interpretations from background theories.

The more common approach [16] for the existing SMT solvers is the integration of a T-solver, i.e. a decision procedure for a given theory T, and a SAT solver. In this model, the SAT solver is in charge of the Boolean formula, while the T-solver analyzes sets of atomic constraints in T. With this, the T-solver checks the possible models the SAT solver generates and rejects them if inconsistencies with the theory appear. In doing so, it gets the efficiency of the SAT solvers for Boolean reasoning, long time tested, and the capability of the more concrete T-oriented algorithms inside the respective theory T.

The current general-use library for SMT is SMT-LIB [3], and there are several implementations of SMT-solvers for it. For our experiments, we use Z3 [7], which implements the theories we need for our purposes:

- QF_LIA (Quantifier Free Linear Integer Arithmetic), which corresponds to quantifier free first order formulas valid in $(\mathbb{Z}, +, -, 0, 1)$,
- QF_LRA (Quantifier Free Linear Real Arithmetic), which corresponds to quantifier free first order formulas valid in $(\mathbb{R}, +, -, \{q : q \in \mathbb{Q}\})$,
- QF_NLRA (Quantifier Free non Linear Real Arithmetic), which corresponds to quantifier free first order formulas valid in $(\mathbb{R}, +, -, \cdot, /, \{q : q \in \mathbb{Q}\})$.

3 A SMT solver for continuous t-norm based fuzzy logics

Inspired by the approach of Ansótegui et. al in [2], we aim at showing in this short paper that a more general solver for fuzzy logics can be implemented using an SMT solver. The main feature of the solver is its *versatility,* so it can be used for testing on a wide range of fuzzy logics (like BL, Łukasiewicz, Gödel, Product and logics obtained through ordinal sums) and also for different kinds of problems, like tautologicity and satisfiability but also logical consequence, or getting evaluations for a given formula (i.e., obtaining variable values that yield a formula a certain truth degree given some restrictions).

It is well-known that every continuous t-norm can be expressed as an ordinal sum of the three main continuous t-norms $*_{Ł}$, $*_G$ and $*_\Pi$. The fact that the three basic t-norms are defined using only addition and multiplications over the real unit interval was used in [2] to develop a solver for theoremhood in these three logics using QF_LRA and QF_NLRA. The case of BL was not considered in [2] because its usual semantics is based on the whole family of continuous t-norms, and not just on a single one. However, thanks to a result of Montagna [11] one can reduce proofs over BL, when working with concrete formulas, to proofs over the logic of an ordinal sum of as many Łukasiewicz components as different variables involved in the set of formulas plus one. This trick is the one we use in our solver for the implementation of BL.

We have implemented a solver that allows the specification, in term of its components, of any continuous t-norm; and the use of BL too. We also allow finitely-valued Łukasiewicz and Gödel logics, and all these logics can be also extended with rational truth-constants. Also, we have considered interesting to add to our software more options than just testing the theoremhood of a formula in a certain logic. In our solver, we can check whether a given formula (possibly with truth-constants) is a logical consequence of a finite set of formulas (possibly with truth-constants as well).

On the other hand, for the particular case of Product logic we have employed a new methodology. This is so because the previous approach, directly coding Product logic connectives with product and division of real numbers, has serious efficiency problems (inherited from QF_NLRA). Indeed, it is already noted in [2] that these problems appear with really simple formulas. To overcome these problems, we have used an alternative coding based on QF_LIA. We can do this because Cignoli and Torrens showed in [5] that the variety of Product algebras is also generated by a discrete linear product algebra: the one with domain the negative cone of the additive group of the integers together with a first element $-\infty$. Indeed, it holds that $TAUT([0,1]_{*_\Pi}) = TAUT((\widetilde{\mathbb{Z}^-})_\oplus)$, where $\widetilde{\mathbb{Z}^-} := \mathbb{Z}^- \cup \{-\infty\}$ endowed with the natural order plus setting $-\infty < x$ for all $x \in \mathbb{Z}^-$, and with its conjunction operation $\oplus$ defined as:

$$x \oplus y := \begin{cases} x + y, \text{ if } x, y \in \mathbb{Z}^- \\ -\infty, \quad \text{otherwise.} \end{cases}$$

Notice that its corresponding residuated implication is then defined as:

$$x \Rightarrow_\oplus y := \begin{cases} 0, & \text{if } x \leq y \\ y - x, & \text{if } x, y \in \mathbb{Z}^-, x > y \\ -\infty, & \text{otherwise.} \end{cases}$$

Therefore, for dealing with Product logic it is enough to consider this discrete algebra; and this particular algebra can be coded using just natural numbers with the addition (i.e. using Presburger Arithmetic). Our experiments have shown that, for concrete instances, this approach based on the discrete theory of integers with the addition (instead of the reals with product) works much better.

In the appendix we present Z3-code examples generated by our software to solve several kind of problems. These examples clarify the methodology explained above.

4 Experimental Results

We consider the main advantage of our solver to be the versatility it allows, but we have also performed an empirical evaluation of our approach using only its theorem-prover option to compare it to [2].

We have conducted experiments over two different families of BL-theorems, see (1) and (2) below. First, for comparison reasons with [2], we have considered the following generalizations (based on powers of the & connective) of the first seven Hájek's axioms of BL [10]:

$$\begin{array}{ll}
(\mathrm{A1}) & (p^n \to q^n) \to ((q^n \to r^n) \to (p^n \to r^n)) \\
(\mathrm{A2}) & (p^n \& q^n) \to p^n \\
(\mathrm{A3}) & (p^n \& q^n) \to (q^n \& p^n) \\
(\mathrm{A4}) & (p^n \& (p^n \to q^n)) \to (q^n \& (q^n \to p^n)) \\
(\mathrm{A5}a) & (p^n \to (q^n \to r^n)) \to ((p^n \& q^n) \to r^n) \\
(\mathrm{A5}b) & ((p^n \& q^n) \to r^n) \to (p^n \to (q^n \to r^n)) \\
(\mathrm{A6}) & ((p^n \to q^n) \to r^n) \to (((q^n \to p^n) \to r^n) \to r^n)
\end{array} \qquad (1)$$

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 [2] the authors refer to [13] to justify why these formulas can be considered a good test bench for (at least) Łukasiewicz logic. In our opinion, these formulas have the drawback of using only three variables. This is a serious drawback at least in Łukasiewicz logic because in this case tautologicity for formulas with three variables can be proved to be solved in polynomial time[3].

To overcome the drawback of the bounded number of variables, we propose a new family of BL-theorems to be used as a bench test. For every $n \in \mathbb{N} \setminus \{0\}$,

$$\bigwedge_{i=1}^{n} (\&_{j=1}^{n}\, p_{ij}) \;\to\; \bigvee_{j=1}^{n} (\&_{i=1}^{n}\, p_{ij}) \qquad (2)$$

[3] This polynomial time result is outside the scope of the present paper, but it can be obtained from the rational triangulation associated with the McNaughton function of the formula with three variables. It is worth noticing that the known proofs of NP-completeness for Łukasiewicz logic [12,10,1] need an arbitrary number of variables.

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 $\big((p_{11}\&p_{12}) \wedge (p_{21}\&p_{22})\big) \rightarrow \big((p_{11}\&p_{21}) \vee (p_{12}\&p_{22})\big)$. We believe these formulas are significantly harder than the ones previously proposed in [13]; and indeed, our experimental results support this claim[4].

4.1 Data results

We ran experiments on a machine with a i5-650 3.20GHz processor and 8GB of RAM. Evaluating the validity in Łukasiewicz, Product and Gödel logics of the generalizations of the BL axioms (1), ranging n from 0 to 500 with increments of 10, throws better results than the ones obtained in [2], but since our solver is an extension of their work for these logics, we suppose this is due to the use of different machines. For Product Logic, we obtained really good timings. 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 [2] 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.

The experiments done with the other family of BL-theorems (2) (see Figure 2 for the results) suggests that here the evaluation time is growing non-polynomially on the parameter n. We have only included in the graph those answers (for parameters $n \leq 70$) obtained in at most 3 hours of execution (e.g. for the BL case we have only got answers 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 modeled with the Presburger arithmetic over $\mathbb{Z}^- \cup \{-\infty\}$ are also smaller than for BL.

5 Conclusions

We have extended the use of SMT technology to define general-use logical solvers for continuous t-norm logics, considered two test suites for these logics, and performed empirical evaluation and testing of our solver. Also, we have provided a new approach for solving more efficiently problems on Product logic.

There are a number of tasks and open questions that we propose as future work. Firstly, solving real applications with SMT-based theorem provers: the

[4] We point out that the natural way to compare our formula with parameter n is to consider the formulas in [13] with the integer part of $\sqrt{n}$ as parameter.

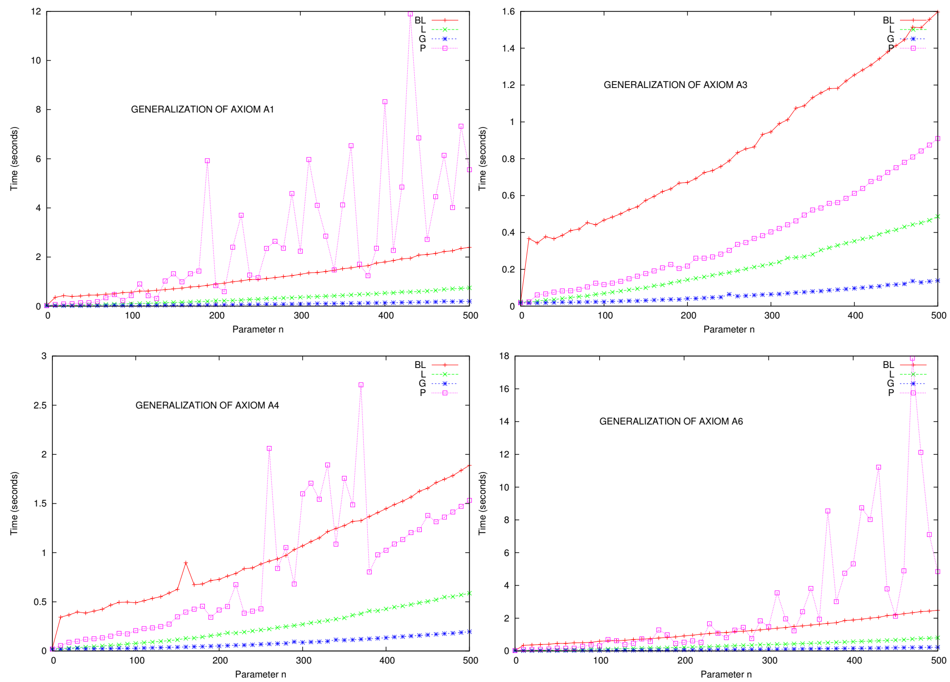

Fig. 1. Generalizations of BL-axioms given in (1).

non-existence of fast and modern theorem provers has limited so far the potential of fuzzy logics to real applications. Secondly, using Presburger arithmetic has been very useful for our solver to deal with product t-norm, but it is still missing an implementation where this trick is used for ordinal sums where one of the components is the product t-norm. Finally, we would like to point out a more challenging problem: to design an SMT solver for MTL logic (i.e., the logic of left-continuous t-norms), since no completeness is currently known using just one particular t-norm.

Acknowledgments. The authors acknowledge support of the Spanish projects TASSAT (TIN2010-20967-C04-01), ARINF (TIN2009-14704-C03-03) and AT (CONSOLIDER CSD2007-0022), and the grants 2009SGR-1433 and 2009SGR-1434 from the Catalan Government. Amanda Vidal is supported by a JAE Predoc fellowship from CSIC.

References

1. S. Aguzzoli, B. Gerla, and Z. Haniková. Complexity issues in basic logic. *Soft Computing*, 9(12):919–934, 2005.
2. C. Ansótegui, M. Bofill, F. Manyà, and M. Villaret. Building automated theorem provers for infinitely valued logics with satisfiability modulo theory solvers. In *ISMVL 2012*, pages 25–30. IEEE Computer Society, 2012.

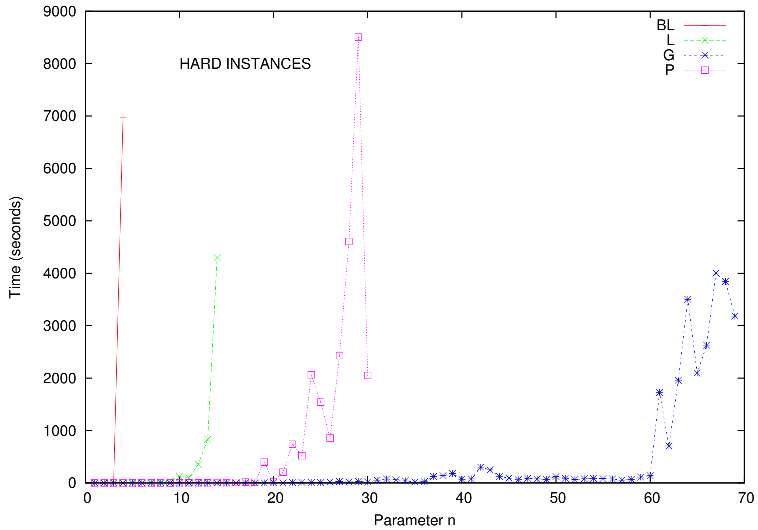

Fig. 2. Our proposed BL-theorems given in (2).

3. 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`.
4. F. Bobillo and U. Straccia. Fuzzy description logics with general t-norms and datatypes. *Fuzzy Sets and Systems*, 160(23):3382–3402, 2009.
5. R. Cignoli and A. Torrens. An algebraic analysis of product logic. *Multiple-valued logic*, 5:45–65, 2000.
6. P. Cintula, P. Hájek, and C. Noguera, editors. *Handbook of Mathematical Fuzzy Logic, 2 volumes*, volume 37 and 38 of *Studies in Logic. Mathematical Logic and Foundation*. College Publications, 2011.
7. L. Mendonça de Moura and N. Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and J. Rehof, editors, *TACAS 2008*, pages 337–340, 2008.
8. F. Esteva, L. Godo, and M. Montagna. Equational Characterization of the Subvarieties of BL Generated by t-norm Algebras. *Studia Logica*, 76(2):161–200, 2004.
9. R. Hähnle. *Automated deduction in multiple valued logics*. Oxford Univ. Pr., 1993.
10. P. Hájek. *Metamathematics of fuzzy logic*. Kluwer Academic Publishers, 1998.
11. F. Montagna. Generating the variety of BL-algebras. *Soft Computing*, 9(12):869–874, 2005.
12. D. Mundici. Satisfiability in many-valued sentential logic is NP-complete. *Theoretical Computer Science*, 52(1-2):145–153, 1987.
13. R. Rothenberg. A class of theorems in Łukasiewicz logic for benchmarking automated theorem provers. In N. Olivetti and C. Schwind, editors, *TABLEAUX '07*, pages 101–111, 2007.
14. S. Schockaert, J. Janssen, and D. Vermeir. Satisfiability checking in Łukasiewicz logic as finite constraint satisfaction. *Journal of Automated Reasoning*. To appear.
15. 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 *LNCS*, pages 240–254. Springer, 2009.

16. R. Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling anc Computation*, 3(3-4):141–224, 2007.
17. A. Vidal, F. Bou, and L. Godo. An SMT-based solver for continuous t-norm based logics. In *Proceedings of SUM 2012*, LNCS. Springer. To appear.

A Appendix

These are a few characteristic examples generated through our software to be run with z3 SMT-solver. It returns $\langle unsat \rangle$ in the validity and logical consequence tests (because these examples are logically valid), and the variables' values in the model generator case.

A.1 Validity of $(x_1 \rightarrow x_2) \rightarrow ((x_2 \rightarrow x_3) \rightarrow (x_1 \rightarrow x_3))$ in the logic with t-norm "$[0, 0.2] \oplus [0.2, 0.7] \oplus [0.7, 0.9] \oplus [0.9, 1]$" where $[0, 0.2]$ and $[0.7, 0.9]$ are $*_{\mathbf{L}}$, and $[0.2, 0.7]$ and $[0.9, 1]$ are $*_G$

```
(set-logic QF_LRA)

; min(x,y)
(define-fun min ((x Real) (y Real)) Real
 (ite (> x y) y x))
; max(x,y)
(define-fun max ((x Real) (y Real)) Real
 (ite (> x y) x y))
; tnorm
(define-fun tnorm ((x Real) (y Real)) Real
 (ite (and (>= x 0) (<= x 0.2)
           (>= y 0) (<= y 0.2))
        (+ 0 (max 0 (- (+ x y) (+ 0 0.2))))
 (ite (and (>= x 0.7) (<= x 0.9)
           (>= y 0.7) (<= y 0.9))
        (+ 0.7 (max 0 (- (+ x y) (+ 0.7 0.9))))
 (ite (and (>= x 0.2) (<= x 0.7)
           (>= y 0.2) (<= y 0.7))
        (min x y)
 (ite (and (>= x 0.9) (<= x 1)
           (>= y 0.9) (<= y 1))
        (min x y)(min x y))))))
;implication (Residuum)
(define-fun impl ((x Real) (y Real)) Real
 (ite (<= x y) 1
 (ite (and (>= x 0) (<= x 0.2)
           (>= y 0) (<= y 0.2))
        (- (+ y 0.2) x)
 (ite (and (>= x 0.7) (<= x 0.9)
           (>= y 0.7) (<= y 0.9))
        (- (+ y 0.9) x)
```

```
 (ite (and (>= x 0.2) (<= x 0.7)
           (>= y 0.2) (<= y 0.7))
        y
 (ite (and (>= x 0.9) (<= x 1)
           (>= y 0.9) (<= y 1))
        y
        y))))))
;negation (not x = x -> 0)
(define-fun neg ((x Real)) Real
 (impl x 0))

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

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

(declare-fun x () Real)
(declare-fun y () Real)
(declare-fun z () Real)
(assert (or
        (and (>= x 0) (<= x 0.2))
        (and (>= x 0.7) (<= x 0.9))
        (and (>= x 0.2) (<= x 0.7))
        (and (>= x 0.9) (<= x 1))))
(assert (or
        (and (>= y 0) (<= y 0.2))
        (and (>= y 0.7) (<= y 0.9))
        (and (>= y 0.2) (<= y 0.7))
        (and (>= y 0.9) (<= y 1))))
(assert (or
        (and (>= z 0) (<= z 0.2))
        (and (>= z 0.7) (<= z 0.9))
        (and (>= z 0.2) (<= z 0.7))
        (and (>= z 0.9) (<= z 1))))
(assert (< (impl (impl x1 x2) (impl (impl x2 x3) (impl x1 x3))) 1) )
(check-sat)
```

A.2 Validy of $(x_1 \to x_2) \to ((x_2 \to x_3) \to (x_1 \to x_3))$ in the Product Logic Modeled over the Presburger Arithmetic

```
(set-logic QF_LIA)

; min(x,y)
(define-fun min ((x Int) (y Int)) Int
 (ite (= x 1) x (ite (= y 1) y (ite (> x y) y x))))
; max(x,y)
(define-fun max ((x Int) (y Int)) Int
```

```
 (ite (= x 1) y (ite (= y 1) x (ite (> x y) x y))))
; tnorm
(define-fun tnorm ((x Int) (y Int)) Int
 (ite (or (= x 1) (= y 1)) 1 (+ x y)))
;implication (Residuum)
(define-fun impl ((x Int) (y Int)) Int
 (ite (= x 1) 0 (ite (= y 1) 1 (ite (<= x y) 0 (- y x)))))
;negation (not x = x -> 0)
(define-fun neg ((x Int)) Int
 (impl x 0))
;conjunction min (x, y)
(define-fun con ((x Int) (y Int)) Int
 (min x y))
;disjunction max (x, y)
(define-fun dis ((x Int) (y Int)) Int
 (max x y))

(declare-fun x () Int)
(declare-fun y () Int)
(declare-fun z () Int)
(assert (< x 2) )
(assert (< y 2) )
(assert (< z 2) )
(assert (or (< (impl (impl x1 x2)
        (impl (impl x2 x3) (impl x1 x3))) 0)
            (= (impl (impl x1 x2)
        (impl (impl x2 x3) (impl x1 x3))) 1)) )
(check-sat)
```

A.3 Validity of (A3) $(p^3\&q^3) \rightarrow (q^3\&p^3)$ in BL

```
(set-logic QF_LRA)

; min(x,y)
(define-fun min ((x Real) (y Real)) Real
 (ite (> x y) y x))
; max(x,y)
(define-fun max ((x Real) (y Real)) Real
 (ite (> x y) x y))
;tnorm
(define-fun tnorm ((x Real) (y Real)) Real
 (ite (and (>= x (/ 0 3)) (<= x (/ 1 3))
           (>= y (/ 0 3)) (<= y (/ 1 3)))
        (+ (/ 0 3) (max 0 (- (+ x y) (+ (/ 0 3) (/ 1 3)))))
 (ite (and (>= x (/ 1 3)) (<= x (/ 2 3))
           (>= y (/ 1 3)) (<= y (/ 2 3)))
        (+ (/ 1 3) (max 0 (- (+ x y) (+ (/ 1 3) (/ 2 3)))))
 (ite (and (>= x (/ 2 3)) (<= x (/ 3 3))
           (>= y (/ 2 3)) (<= y (/ 3 3)))
        (+ (/ 2 3) (max 0 (- (+ x y) (+ (/ 2 3) (/ 3 3)))))
```

```
        (min x y)))))
;implication (Residuum)
(define-fun impl ((x Real) (y Real)) Real
(ite (<= x y) 1
 (ite (and (>= x (/ 0 3)) (<= x (/ 1 3))
           (>= y (/ 0 3)) (<= y (/ 1 3)))
        (- (+ y (/ 1 3)) x)
 (ite (and (>= x (/ 1 3)) (<= x (/ 2 3))
           (>= y (/ 1 3)) (<= y (/ 2 3)))
        (- (+ y (/ 2 3)) x)
 (ite (and (>= x (/ 2 3)) (<= x (/ 3 3))
           (>= y (/ 2 3)) (<= y (/ 3 3)))
        (- (+ y (/ 3 3)) x)
        y)))))
;negation (not x = x -> 0)
(define-fun neg ((x Real)) Real
 (impl x 0))
;conjunction min (x, y)
(define-fun con ((x Real) (y Real)) Real
 (min x y))
;disjunction max (x, y)
(define-fun dis ((x Real) (y Real)) Real
(max x y))
(declare-fun x () Real)
(declare-fun y () Real)
(assert (and (>= x 0) (<= x 1)) )
(assert (and (>= y 0) (<= y 1)) )
(assert (< (impl (tnorm (tnorm x x) (tnorm y y))
                 (tnorm (tnorm y y) (tnorm x x))) 1) )
(check-sat)
```

A.4 Logical consequence over Łukasiewicz for $x_1 \to x_2,\ x_2 \to x_3 \models x_1 \to x_3$

```
(set-logic QF_LRA)

; min(x,y)
(define-fun min ((x Real) (y Real)) Real
 (ite (> x y) y x))
; max(x,y)
(define-fun max ((x Real) (y Real)) Real
 (ite (> x y) x y))
;tnorm
(define-fun tnorm ((x Real) (y Real)) Real
 (ite (and (>= x 0) (<= x 1)
           (>= y 0) (<= y 1))
        (+ 0 (max 0 (- (+ x y) (+ 0 1))))
        (min x y)))
;implication (Residuum)
(define-fun impl ((x Real) (y Real)) Real
```

```
 (ite (<= x y) 1
 (ite (and (>= x 0) (<= x 1)
           (>= y 0) (<= y 1))
        (- (+ y 1) x)
        y)))
;negation (not x = x -> 0)
(define-fun neg ((x Real)) Real
 (impl x 0))
;conjunction min (x, y)
(define-fun con ((x Real) (y Real)) Real
 (min x y))
;disjunction max (x, y)
(define-fun dis ((x Real) (y Real)) Real
 (max x y))

(declare-fun x1 () Real)
(declare-fun x2 () Real)
(declare-fun x3 () Real)
(assert (and (>= x1 0) (<= x1 1))  )
(assert (and (>= x2 0) (<= x2 1))  )
(assert (and (>= x3 0) (<= x3 1))  )
(assert (= (impl x1 x2) 1) )
(assert (= (impl x2 x3) 1) )
(assert (< (impl x1 x3) 1) )
(check-sat)
```

A.5 Satisfiability testing and model generation on Łukasiewicz of the set of formulas $\{v_0 * v_0 = 0.5,\ v_1 * v_1 = v_0,\ v_2 * v_2 = v_1\}$

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

;....Lukasiewicz connectives definitions (see Appendix A.4)

(declare-fun v0 () Real)
(declare-fun v1 () Real)
(declare-fun v2 () Real)
(assert (<= v0 1) )
(assert (>= v0 0) )
(assert (<= v1 1) )
(assert (>= v1 0) )
(assert (<= v2 1) )
(assert (>= v2 0) )
(assert (= (tnorm v0 v0) 0.5) )
(assert (= (tnorm v1 v1) v0) )
(assert (= (tnorm v2 v2) v1) )
(check-sat)
(get-value (v0))
(get-value (v1))
(get-value (v2))
```