Some Remarks Regarding MTL and Divisible Residuated Algebras

Cristina Flaut *, Dana Piciu, Radu Vasile

Posted Date: 9 October 2025

doi: 10.20944/preprints202510.0145.v1

Keywords: (divisible) residuated lattice; MTL-algebra; ring

 doi:10.20944/preprints202510.0145.v1

Article

Some Remarks Regarding MTL and Divisible Residuated Algebras

Cristina Flaut [1,*], Dana Piciu [2] and Radu Vasile [1]

[1] Faculty of Mathematics and Computer Science, Ovidius University, Bd. Mamaia 124, 900527, Constanţa, România
[2] Faculty of Science, University of Craiova, A.I. Cuza Street, 13, 200585, Craiova, Romania
* Correspondence: cflaut@univ-ovidius.ro

Abstract

Divisible residuated lattices and MTL-algebras are algebraic structures connected with algebras in t-norm based fuzzy logics, being examples of BL-algebras. They are an important significance in the study of fuzzy logic. The purpose of this paper is to investigate and give classifications of these types of algebras. From computational considerations, we analyze the structure of these residuated lattices of small size n ($2 \leq n \leq 5$) and we give summarizing statistics. To extend these results for higer size, we used computer and a constructive algorithm for generating all residuated lattices.

Keywords: (divisible) residuated lattice; MTL-algebra; ring

---

1. Introduction

Nonclassical logics have been represented as algebras and they are related to computer science. Residuated lattices were introduced by Dilworth and Ward (see [Di; 38], [WD; 39]) and their study is originated in the context of theory of rings. It is known that the lattice of ideals in a commutative ring is a residuated lattice, see [Bl; 53].

Divisible residuated lattices (or residuated lattice ordered monoids or RL-monoids) and MTL-algebras are two examples of residuated lattices.

Divisible residuated lattices were introduced in [S] as a unifying concept for Heyting algebras and Abelian lattice ordered groups. These residuated lattices satisfying divisibility condition are connected with algebras in t-norm based fuzzy logics, being examples of BL-algebras.

Esteva et al. introduced in [E] the t-norm -based fuzzy logic MTL, to capture the logic of t-norm-based fuzzy logics and their residuum, The resulting algebras are MTL-algebras.

Since BL-algebras are MTL-algebras satisfying divisibility property the question that one can ask is how many residuated lattices satisfy one property and how many satisfy the other.

Properties of MTL-algebras, divisible residuated lattices and a classification of finite residuated lattices can be found in several papers, as for example: [BV; 10], [CFP; 23], [FP; 22], [FP; 23].

In this paper, we present a characterisation for residuated lattices of size $n \leq 4$ and we construct all (up to an isomorphism) divisible residuated lattices and MTL-algebras with $n \leq 5$ elements. Also, we present summarizing statistics. This method can be used to construct finite residuated lattices of larger size, the inconvenience being the large number of algebras that should be generated. This is the reason for which we consider a better idea to use computer. Differences between our approach and the classification given in [BV; 10] are that we give a mathematical proof of such a construction, by using ordinal product, and we provide a constructive algorithm for generating all residuated lattices, which can be easy addapted for higer sizes.

2. Preliminaries

Definition 1 ([Di; 38], [WD; 39]) A *(commutative) residuated lattice* is an algebra $(L, \wedge, \vee, \odot, \rightarrow, 0, 1)$ with an order $\leq$ such that:

 doi:10.20944/preprints202510.0145.v1

(i) $(L, \wedge, \vee, 0, 1)$ is a bounded lattice;
(ii) $(L, \odot, 1)$ is a commutative ordered monoid;
(iii) $z \leq x \to y$ iff $x \odot z \leq y$, for all $x, y, z \in L$.
If L is a residuated lattice, then we denote $x^* = x \to 0$, for any $x \in L$. .
In a residuated lattice L we consider the identities:

$$(prel) \qquad (x \to y) \vee (y \to x) = 1 \qquad (prelinearity);$$

$$(div) \qquad x \odot (x \to y) = x \wedge y \qquad (divisibility).$$

Definition 2 ([COM; 09, [CHA; 58], [NL; 03], [I; 09], [T; 99]) A residuated lattice L is called:
(i) *an MTL-algebra* if L verifies $(prel)$ condition;
(ii) *divisible* if L verifies (div) condition;
(iii) *a BL-algebra* if L verifies $(prel) + (div)$ conditions. A *BL-chain* is a totally ordered BL-algebra.
(iv) *an MV-algebra* is a BL-algebra in which $x^{**} = x$, for every $x \in L$.

Example 3. 1) For a commutative unitary ring A, the lattice of ideals $(Id(A), \cap, +, \otimes \to, 0 = \{0\}, 1 = A)$ is a residuated lattice with the order relation is $\subseteq$ and

$$I + J =< I \cup J >= \{i + j, i \in I, j \in J\},$$

$$I \otimes J = \{\sum_{k=1}^{n} i_k j_k,\ i_k \in I, j_k \in J\},$$

$$I \to J = (J : I) = \{x \in A, x \cdot I \subseteq J\},$$

$$Ann(I) = (\mathbf{0} : I), \text{ where } \mathbf{0} =< 0 >,$$

for every $I, J \in Id(A)$, see [TT; 22]. Moreover, $I + J$, $I \otimes J$, $I \to J$ and $Ann(I)$ are ideals of A, called sum, product, quotient and annihilator, see [BP; 02].

2) If $\mathcal{L}_1 = (L_1, \wedge_1, \vee_1, \odot_1, \to_1, 0_1, 1_1)$ and $\mathcal{L}_2 = (L_2, \wedge_2, \vee_2, \odot_2, \to_2, 0_2, 1_2)$ are two residuated lattices such that $1_1 = 0_2$ and $(L_1 \backslash \{1_1\}) \cap (L_2 \backslash \{0_2\}) = \emptyset$, then, the ordinal product (sum) of $\mathcal{L}_1$ and $\mathcal{L}_2$ is the residuated lattice $\mathcal{L}_1 \boxtimes \mathcal{L}_2 = (L_1 \cup L_2, \wedge, \vee, \odot, \to, 0, 1)$ where

$$0 = 0_1 \text{ and } 1 = 1_2,$$

$$x \leq y \text{ if } (x, y \in L_1 \text{ and } x \leq_1 y) \text{ or } (x, y \in L_2 \text{ and } x \leq_2 y) \text{ or } (x \in L_1 \text{ and } y \in L_2),$$

$$x \to y = \begin{cases} 1, \text{ if } x \leq y, \\ x \to_i y, \text{ if } x \not\leq y,\ x, y \in L_i,\ i = 1, 2, \\ y, \text{ if } x \not\leq y,\ x \in L_2,\ y \in L_1 \backslash \{1_1\}. \end{cases}$$

$$x \odot y = \begin{cases} x \odot_1 y, \text{ if } x, y \in L_1, \\ x \odot_2 y, \text{ if } x, y \in L_2, \\ x, \text{ if } x \in L_1 \backslash \{1_1\} \text{ and } y \in L_2. \end{cases},$$

see [I; 09].

3) For $n \geq 1$, we consider the chain $F_{n+1} = \{0, 1, 2, ..., n\}$ organized as lattice by $\wedge = \min$, $\vee = \max$ and as bounded residuated lattice $\mathcal{F}o_{n+1} = (F_{n+1}, \min, \max, \odot_F, \to_F, 0, n)$ with de Fodor's implication $\to_F$:

$$x \to_F y = \begin{cases} n, \text{ if } x \leq y, \\ \max(n - x, y), \text{ if } x > y, \end{cases}$$

 doi:10.20944/preprints202510.0145.v1

with the corresponding Fodor's t-norm $\odot_F$:

$$x \odot_F y = \begin{cases} 0, \text{ if } x \leq n - y, \\ \min(x, y), \text{ if } x > n - y \end{cases},$$

see [F].

3. Main Results

Proposition 4. ([I; 09]) Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be residuated lattices.
(i) If $\mathcal{L}_1$ and $\mathcal{L}_2$ are divisible, then $\mathcal{L}_1 \boxtimes \mathcal{L}_2$ *is divisible;*
(ii) If $\mathcal{L}_1$ and $\mathcal{L}_2$ are MTL-algebras and $\mathcal{L}_1$ is a chain, then $\mathcal{L}_1 \boxtimes \mathcal{L}_2$ *is an MTL-algebra.*

Remark 5. 1) If $\mathcal{L}_1$ and $\mathcal{L}_2$ are BL-algebras and $\mathcal{L}_1$ is a chain, then $\mathcal{L}_1 \boxtimes \mathcal{L}_2$ *is a BL-algebra.*
2) If $\mathcal{L}_1$ and $\mathcal{L}_2$ are BL-algebras and $\mathcal{L}_1$ is not a chain, then $\mathcal{L}_1 \boxtimes \mathcal{L}_2$ *is a divisible residuated lattice which is not a BL-algebra, since it is not an MTL-algebra.*

In the following, we present some examples of divisible and MTL algebras and an way to obtain its.

Example 6. To generate a divisible residuated lattice with $k + 4$ elements, $k \geq 1$, which is not an MTL-algebra organized as lattice as in Figure 1,

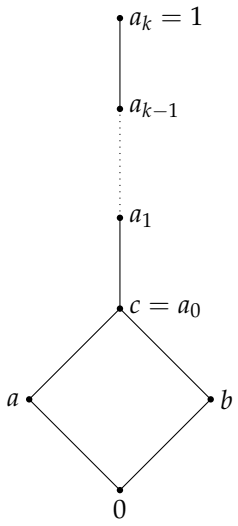


Figure 1.

We consider the commutative rings $(\mathbb{Z}_{2^k}, +, \cdot)$ and $(\mathbb{Z}_2 \times \mathbb{Z}_2, +, \cdot)$.
For $\mathbb{Z}_2 \times \mathbb{Z}_2$ the lattice of ideals is $Id(\mathbb{Z}_2 \times \mathbb{Z}_2) = \{\left(\widehat{0}, \widehat{0}\right), \{\left(\widehat{0}, \widehat{0}\right), \left(\widehat{0}, \widehat{1}\right)\}, \{\left(\widehat{0}, \widehat{0}\right), \left(\widehat{1}, \widehat{0}\right)\}, \mathbb{Z}_2 \times \mathbb{Z}_2\} = \{O, A, B, E\}$, which is a Boolean algebra $(Id(\mathbb{Z}_2 \times \mathbb{Z}_2), \cap, +, \otimes \to, 0 = \{\left(\widehat{0}, \widehat{0}\right)\}, 1 = \mathbb{Z}_2 \times \mathbb{Z}_2)$, so a BL-algebra, with the following operations:

$\to$	O	A	B	C
ine O	C	C	C	C
A	B	C	B	C
B	A	A	C	C
C	O	A	B	C

and

$\otimes$	O	A	B	C
ine O	O	O	O	O
A	O	A	O	A
B	O	O	B	B
C	O	A	B	C

.

Also, we know that $(Id(Z_{2^k}), \cap, +, \otimes, \to, 0 = \{0\}, 1 = Z_{2^k})$ is the only MV-chain (up to an isomorphism) with $k + 1$ elements, see [FP; 22], so, it is a BL-chain.

The ring $(Z_{2^k}, +, \cdot)$ has $k+1$ ideals: $I_0 = \{0\}$, $I_1 = \widehat{2^{k-1}}Z_{2^k}$, ..., $I_{k-2} = \widehat{2^2}Z_{2^k}$, $I_{k-1} = \widehat{2}Z_{p^k}$, $I_k = Z_{2^k}$ and $I_0 \subseteq I_1 \subseteq I_2 \subseteq ... \subseteq I_k$.

 doi:10.20944/preprints202510.0145.v1

For every $i, j \in \{0, ..., k\}$ we have

$$I_i \to I_j = Z_{2^k} \text{ if } i \leq j \text{ and } I_{k-i+j} \text{ otherwise}$$

and

$$I_i \otimes I_j = I_0 \text{ if } i + j \leq k \text{ and } I_{i+j-k} \text{ otherwise.}$$

We consider two BL-algebras isomorphic with $(Id(\mathbb{Z}_2 \times \mathbb{Z}_2), \cap, +, \otimes \to, 0 = \{\left(\widehat{0}, \widehat{0}\right)\}, 1 = \mathbb{Z}_2 \times \mathbb{Z}_2)$ and $(Id(\mathbb{Z}_{2^k}), \cap, +, \otimes \to, 0 = \{0\}, 1 = \mathbb{Z}_{2^k})$ denoted by $\mathcal{L}_1 = (L_1 = \{0, a, b, c = a_0\}, \wedge_1, \vee_1, \odot_1, \to_1, 0, c = a_0)$, and $\mathcal{L}_2 = (L_2 = \{c = a_0, a_1, ...a_{k-1}, a_k\}, \wedge_2, \vee_2, \odot_2, \to_2, 0, a_k = 1)$. Using Proposition 4, we generate a divisible residuated lattice $\mathcal{L}_1 \boxtimes \mathcal{L}_2 = (L_1 \cup L_2 = \{0, a, b, c = a_0, a_1, ...a_{k-1}, a_k = 1\}, \wedge, \vee, \odot, \to, 0, 1)$ with $k + 4$ elements, for any $k \geq 1$.

This residuated lattice is not an MTL algebra since $(a \to b) \vee (b \to a) = b \vee a = c \neq 1$.

For example, for $k = 4$ we obtain a divisible residuated lattice **(**which is not a MTL algebra**)** $\mathcal{L}_1 \boxtimes \mathcal{L}_2 = (\{0, a, b, c = a_0, a_1, a_2, a_3, a_4 = 1, \}, \wedge, \vee, \odot, \to, 0, 1)$ with the following operations:

$\to$	0	a	b	$c = a_0$	a_1	a_2	a_3	$a_4 = 1$
ine 0	1	1	1	1	1	1	1	1
a	b	1	b	1	1	1	1	1
b	a	a	1	1	1	1	1	1
$c = a_0$	0	a	b	1	1	1	1	1
a_1	0	a	b	a_3	1	1	1	1
a_2	0	a	b	a_2	a_3	1	1	1
a_3	0	a	b	a_1	a_2	a_3	1	1
$a_4 = 1$	0	a	b	c	a_1	a_2	a_3	1

and

$\odot$	0	a	b	$c = a_0$	a_1	a_2	a_3	$a_4 = 1$
ine 0	0	0	0	0	0	0	0	0
a	0	a	0	a	a	a	a	a
b	0	0	b	b	b	b	b	b
$c = a_0$	0	a	b	c	c	c	c	c
a_1	0	a	b	c	c	c	c	a_1
a_2	0	a	b	c	c	c	a_1	a_2
a_3	0	a	b	c	c	a_1	a_2	a_3
$a_4 = 1$	0	a	b	c	a_1	a_2	a_3	1

since in $(Id(Z_{2^4}), \cap, +, \otimes, \to, 0 = \{0\}, 1 = Z_{2^4})$, we have

$$I_i \to I_j = I_4 \text{ if } i \leq j \text{ and } I_{4-i+j} \text{ if } i > j$$

and

$$I_i \otimes I_j = I_0 \text{ if } i + j \leq 4 \text{ and } I_{i+j-4} \text{ if } i + j > 4,$$

so $Id(Z_{2^4})$ is a BL-algebra with 5 elements with the operations:

$\to$	I_0	I_1	I_2	I_3	I_4
$ine I_0$	I_4	I_4	I_4	I_4	I_4
I_1	I_3	I_4	I_4	I_4	I_4
I_2	I_2	I_3	I_4	I_4	I_4
I_3	I_1	I_2	I_3	I_4	I_4
I_4	I_0	I_1	I_2	I_3	I_4

,

$\otimes$	I_0	I_1	I_2	I_3	I_4
$ine I_0$	I_0	I_0	I_0	I_0	I_0
I_1	I_0	I_0	I_0	I_0	I_1
I_2	I_0	I_0	I_0	I_1	I_2
I_3	I_0	I_0	I_1	I_2	I_3
I_4	I_0	I_1	I_2	I_3	I_4

.

 doi:10.20944/preprints202510.0145.v1

Examples 7. To generate an MTL-algebra with 8 elements (which is not divisible) organized as a lattice as in Figure 2,

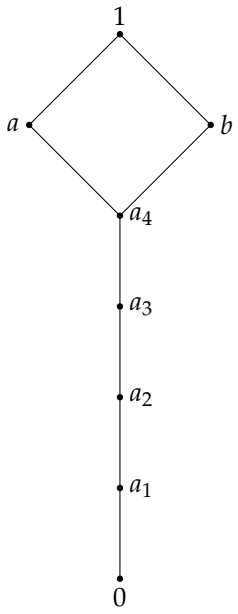


Figure 2.

we consider the MTL-algebra $\mathcal{L}_1 = (L_1 = \{0 \leq a_1 \leq a_2 \leq a_3 \leq a_4\}, \wedge_1, \vee_1, \odot_1, \to_1, 0, a_4)$ with the following operations:

$\to_1$	0	a_1	a_2	a_3	a_4
$ine0$	a_4	a_4	a_4	a_4	a_4
a_1	0	a_4	a_4	a_4	a_4
a_2	0	a_3	a_4	a_4	a_4
a_3	0	a_3	a_3	a_4	a_4
a_4	0	a_1	a_2	a_3	a_4

,

$\odot_1$	0	a_1	a_2	a_3	a_4
$ine0$	0	0	0	0	0
a_1	0	a_1	a_1	a_1	a_1
a_2	0	a_1	a_1	a_1	a_2
a_3	0	a_1	a_1	a_1	a_3
a_4	0	a_1	a_2	a_3	a_4

,

(see [I; 09], p.218) and BL-algebra $\mathcal{L}_2 = (L_2 = \{a_4, a, b, 1\}, \wedge_2, \vee_2, \odot_2, \to_2, a_4, 1)$ isomorphic with $(Id(\mathbb{Z}_2 \times \mathbb{Z}_2), \cap, +, \otimes \to, 0 = \{\left(\widehat{0}, \widehat{0}\right)\}, 1 = \mathbb{Z}_2 \times \mathbb{Z}_2)$.

Then $\mathcal{L}_1 \boxtimes \mathcal{L}_2 = (L_1 \cup L_2 = \{0, a_1, a_1, a_3, a_4, a, b, 1\}, \wedge, \vee, \odot, \to, 0, 1)$ is an MTL-algebra, by Proposition 4, with the operations:

$\to$	0	a_1	a_2	a_3	a_4	a	b	1
ine 0	1	1	1	1	1	1	1	1
a_1	0	1	1	1	1	1	1	1
a_2	0	a_3	1	1	1	1	1	1
a_3	0	a_3	a_3	1	1	1	1	1
a_4	0	a_1	a_2	a_3	1	1	1	1
a	0	a_1	a_2	a_3	b	1	b	1
b	0	a_1	a_2	a_3	a	a	1	1
1	0	a_1	a_2	a_3	a_4	a	b	1

and

$\odot$	0	a_1	a_2	a_3	a_4	a	b	1
ine 0	0	0	0	0	0	0	0	0
a_1	0	a_1	a_1	a_1	a_1	a_1	a_1	a_1
a_2	0	a_1	a_1	a_1	a_2	a_2	a_2	a_2
a_3	0	a_1	a_1	a_1	a_3	a_3	a_3	a_3
a_4	0	a_1	a_2	a_3	a_4	a_4	a_4	a_4
a	0	a_1	a_2	a_3	a_4	a	a_4	a
b	0	a_1	a_2	a_3	a_4	a_4	b	b
1	0	a_1	a_2	a_3	a_4	a	b	1

.

Remark 8. Let L be a residuated lattice with 2 elements. Obviously, $L = \{0, 1\}$ with

$\to$	0	1
0	1	1
1	0	1

and

$\odot$	0	1
0	0	0
1	0	1

.

 doi:10.20944/preprints202510.0145.v1

Obviously, L is the Boolean algebra L_2, thus L has (div) and $(prel)$ properties.

Remark 9. Let L be a residuated lattice with 3 elements, so $L = \{0, a, 1\}$. Obviously, $0 \leq a \leq 1$ and L is a chain.

We have the following cases:

1) $a^2 = a$. Obviously, $a^* \neq a$. If $a^* = a \Rightarrow 0 = a \odot a^* = a^2 = a$, a contradiction.

If $a^* = 0$ we obtain a residuated lattice $L_1 = \{0, a, 1\}$ with the following operations:

$\rightarrow$	0	a	1	$\odot$	0	a	1
ine 0	1	1	1	ine 0	0	0	0
a	0	1	1	a	0	a	a
1	0	a	1	1	0	a	1

.

L_1 is a BL-algebra isomorphic with $Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_2)$, that is not an MV-algebra.

Obviously, L_1 satisfies (div)+$(prel)$ properties.

If $a^* = 1 \Rightarrow 0 = a \odot a^* = a$, a contradiction.

2) $a^2 = 0$. In this case, $a \leq a^*$, so, $a^* = a, 1$.

If $a^* = a$ we obtain a residuated lattice $L_2 = \{0, a, 1\}$ with the following operations:

$\rightarrow$	0	a	1	$\odot$	0	a	1
ine 0	1	1	1	ine 0	0	0	0
a	a	1	1	a	0	0	a
1	0	a	1	1	0	a	1

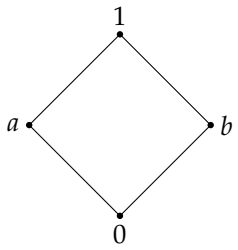
.

L_2 is an MV-algebra isomorphic with $\left(Id\left(\mathbb{Z}_{p^2}\right), +, \cdot\right)$, $p \geq 2$ a prime number.

Obviously, L_2 satisfies (div)+$(prel)$ properties.

If $a^* = 1 \Rightarrow 0 = a \odot a^* = a \odot 1 = a$, a contradiction.

Remark 10. Let L be a residuated lattice with 4 elements, $L = \{0, a, b, 1\}$. Then, the lattice L can be of the form A or B from Figure 3.



A

B

Figure 3.

Case i. a and b are incomparable elements, as in Figure 3, A.

 doi:10.20944/preprints202510.0145.v1

In this situation, $a \vee b = 1$ and $a \wedge b = 0$, so, $a \odot b = 0$. We deduce that $a \leq b^*$, so $b^* = a$. Indeed, if $b^* = 1 \Rightarrow 0 = b \odot b^* = b \odot 1 = b$, a contradiction.

Analogously, $a^* = b$.

Moreover, $a^2 = a$ and $b^2 = b$. Indeed, $a^2 \leq a$ and if $a^2 = 0$ then $a \leq a^* = b$, a contradiction.

Also, $b \leq a \to b$, so $a \to b = b$ since $a \to b = 1$ implies $a \leq b$, a contradiction. It is clear that $b \to a = a$. Therefore, we obtain a Boolean algebra $L_4 = \{0, a, b, 1\}$ with the following operations:

$\to$	0	a	b	1
ine 0	1	1	1	1
a	b	1	b	1
b	a	a	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	a	0	a
b	0	0	b	b
1	0	a	b	1

izomorphic with $Id(\mathbb{Z}_2 \times \mathbb{Z}_2)$. Thus L_4 satisfies (div)+$(prel)$ properties.

Case ii. The lattice is a chain, $0 \leq a \leq b \leq 1$, as in Figure 3, B.

We have the following subcases:

1) $a^2 = 0, b^2 = a, a \odot b = 0$. We have $a^* = b$ and $b^* = a$. Also, $b \to a = b$. Therefore, we obtain an MV-algebra structure $L_3 = \{0, a, b, 1\}$ with the following operations:

$\to$	0	a	b	1
ine 0	1	1	1	1
a	b	1	1	1
b	a	b	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	0	0	a
b	0	0	a	b
1	0	a	b	1

.

We remark that L_3 is isomorphic with $Id(\mathbb{Z}_8)$ and satisfies (div)+$(prel)$ properties.

2) $a^2 = 0, b^2 = a, a \odot b = a$. In this case we do not obtaine a residuated lattice, since $\odot$ is not associative. For example, $a \odot b^2 = a^2 = 0$ and $(a \odot b) \odot b = a \odot b = a$.

3) $a^2 = 0, b^2 = b, a \odot b = a$. We have $a^* = a$ and $b^* = 0$. Also, $b \to a = a$. Therefore, we obtain a BL-algebra structure $L_5 = \{0, a, b, 1\}$ (which is not an MV-algebra) with the following tables:

$\to$	0	a	b	1
ine 0	1	1	1	1
a	a	1	1	1
b	0	a	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	0	a	a
b	0	a	b	b
1	0	a	b	1

.

We remark that L_5 is isomorphic with $Id(\mathbb{Z}_4) \boxtimes Id(\mathbb{Z}_2)$ and satisfies $(div) + (prel)$ properties.

4) $a^2 = 0, b^2 = b, a \odot b = 0$. We have $a^* = b$ and $b^* = a$. Also, $b \to a = a$. Therefore, we have $b \odot (b \to a) = b \odot a = 0$ and $b \wedge a = a$, so, condition (div) is not satisfied. It results that $L_6 = \{0, a, b, 1\}$ is not a BL-algebra, so it is only an MTL-algebra with the following tables:

$\to$	0	a	b	1
ine 0	1	1	1	1
a	b	1	1	1
b	a	a	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	0	0	a
b	0	0	b	b
1	0	a	b	1

.

We remark that L_6 is isomorphic with $\mathcal{F}o_4$ and satisfies $(prel)$ property and do not satisfies (div) property.

5) $a^2 = b^2 = a \odot b = 0$. We have $a^* = b^* = b$. Also, $b \to a = b$. Therefore, we have $b \odot (b \to a) = b \odot b = 0$ and $b \wedge a = a$, false. Condition (div) is not satisfied. It results that $L_7 = \{0, a, b, 1\}$ is not a BL-algebra. It is only an MTL-algebra with the following tables:

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 9 October 2025 doi:10.20944/preprints202510.0145.v1

$\rightarrow$	0	a	b	1
ine 0	1	1	1	1
a	b	1	1	1
b	b	b	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	0	0	a
b	0	0	0	b
1	0	a	b	1

.

We remark that L_7 is isomorphic with the residuated lattice $\mathcal{A}(a,b)$, see [I; 09], p. 221, and satisfies $(prel)$ property and do not satisfies (div) property.

6) $a^2 = b^2 = 0, a \odot b = a$, it is not possible, since $a = a \odot b < b^2 = 0$.

7) $a^2 = b^2 = a \odot b = a$. We have $a^* = b^* = 0$. Also, $b \rightarrow a = b$. Therefore, for $L_8 = \{0, a, b, 1\}$ we obtain a BL-algebra structure (which is not an MV-algebra), isomorphic with $Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_4)$ with the following tables:

$\rightarrow$	0	a	b	1
ine 0	1	1	1	1
a	0	1	1	1
b	0	b	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	a	a	a
b	0	a	a	b
1	0	a	b	1

.

We remark that L_8 satisfies (div) and $(prel)$ properties.

8) $a^2 = b^2 = a, a \odot b = 0$ is not possible, since $a = a^2 \leq a \odot b = 0$.

9) $a^2 = a, b^2 = b, a \odot b = a$. We have $a^* = b^* = 0$ and $b \rightarrow a = a$. Therefore, for $L_9 = \{0, a, b, 1\}$ we obtain a BL-algebra structure (which is not an MV-algebra) with the following tables:

$\rightarrow$	0	a	b	1
ine 0	1	1	1	1
a	0	1	1	1
b	0	a	1	1
1	0	a	b	1

and

$\odot$	0	a	b	1
ine 0	0	0	0	0
a	0	a	a	a
b	0	a	b	b
1	0	a	b	1

.

We remark that L_9 is isomorphic with $Id(\mathbb{Z}_2) \boxtimes (Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_2))$ and satisfies $(div) + (prel)$ properties.

10) $a^2 = a, b^2 = b, a \odot b = 0$ is not possible, since $a = a^2 \leq a \odot b = 0$.

11) $a^2 = a, b^2 = 0, a \odot b = a$ is not possible, since $a = a \odot b \leq b^2 = 0$.

10) $a^2 = a, b^2 = a \odot b = 0$, is not possible, since from $a \leq b$ we obtain $a = a^2 \leq b^2 = 0$, a contradiction.

Counting the BL-algebras, MTL-algebras and divisible residuated lattices of order 4 we obtain the following result:

Proposition 11. *There are 7 residuated lattices of order 4: 2 MV-algebeas 5 BL-algebras, 5 divisible residuated lattices and 7 MTL algebras.*

Theorem 12. *i) All residuated lattices with $n \leq 4$ elements are MTL-algebras.*

ii) There are no divisible residuated lattices with $n \leq 4$ elements that are not MTL-algebras.

In **Table 1,** we briefly describe the structure of finite residuated lattices L with $2 \leq n \leq 4$ elements:

 doi:10.20944/preprints202510.0145.v1

Table 1.

$\|L\|=$ **n**	**Nr of residuated lattices**	**Structure**
$n=2$	1	$\{Id(\mathbb{Z}_2)$ (BL-chain)
$n=3$	2	$\begin{cases} Id(\mathbb{Z}_4) \text{ (BL-chain)} \\ Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_2) \text{ (BL-chain)} \end{cases}$
$n=4$	7	$\begin{cases} Id(\mathbb{Z}_8) \text{ (BL-chain)} \\ Id(\mathbb{Z}_2 \times \mathbb{Z}_2) \text{ (BL, not chain)} \\ Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_4) \text{ (BL-chain)} \\ Id(\mathbb{Z}_4) \boxtimes Id(\mathbb{Z}_2) \text{ (BL-chain)} \\ Id(\mathbb{Z}_2) \boxtimes (Id(\mathbb{Z}_2) \boxtimes Id(\mathbb{Z}_2)) \text{ (BL-chain)} \\ \mathcal{F}o_4 \text{ (MTL-chain, not BL)} \\ \mathcal{A}(a,b) \text{ (MTL-chain, not BL).} \end{cases}$

Table 2 present a summary for the number of all residuated lattices, Boolean algebras, MV-algebras, BL-algebras, MTL-algebras and divisible residuated lattices with $n \leq 4$ elements.

Table 2.

	$n=2$	$n=3$	$n=4$
All res. latt.	**1**	**2**	**7**
Boole	**1**	**-**	**1**
MV	**1**	**1**	**2**
BL	**1**	**2**	**5**
DIV	**1**	**2**	**5**
MTL	**1**	**2**	**7**

4. The Basic Algorithm and the Made Improvements

In this section, we present a constructive algorithm for generating all residuated lattices and all made improvements which allow us to extend it to higer sizes. The initial algorithm exhaustively traverses the execution space of possible structures, verifying the axioms given in the definition of residuated lattices. Since, in this case, the execution time is very high, we improved this algorithm and we present a set of algorithmic optimizations that significantly reduce the execution space and execution time. At the end, we present other possible optimizations to achieve higher orders for n, where n is the size of a residuated lattice.

The initial algorithm exhaustively traverses the search space of possible structures, verifying the basic axioms of the residuated networks and the mentioned properties. Next, we present a set of algorithmic optimizations that significantly reduce the search space and execution time. Finally, we present other possible optimizations to achieve higher orders of n: (1) canonicalization of the order relation to avoid repeated generation of isomorphic structures, (2) intelligent pruning of backtracking for the $\otimes$ operation based on lower/upper bounds and local checks (partial monotonicity and right distributivity), and (3) efficient isomorphic deduplication using the network automorphism group. We discuss the benefits of each optimization and current limitations, as well as possible directions for further improvement.

5. Basic Algorithm

The constructive algorithm systematically explores all candidate structures, going through the following essential steps:

1. Generation of posets (partially ordered sets) with 0 **and** 1 ***fixed.*** First, all possible order relations (posets) on the set $L = \{0, \ldots, n-1\}$ that have 0 as the minimum element and 1 as the maximum element are generated. The representation of the relation $\leq$ is done by its transitively closed adjacency matrix (a Boolean matrix $n \times n$). Specifically, we examine each subset of comparable pairs (i, j) with $i < j$ as possible edges in the graph, compute the transitive closure (using the Roy–Warshall algorithm), and then filter out the results that are not antisymmetric (i.e., remove graphs containing

 doi:10.20944/preprints202510.0145.v1

cycles $i \leq j$ and $j \leq i$ with $i \neq j$). The algorithm ensures that 0 remains the smallest element (i.e., $0 \leq x$ for any x) and 1 the largest ($x \leq 1$ for any x). The code snippet below illustrates the generation of all posets labeled with 0 minimum and 1 maximum, using the Floyd–Warshall algorithm for transitivity:

```
\{Floyd--Warshall on Boolean relations and the generation of posets labeled
with \$0\$ minimum and \$1\$ maximum.\}
def floyd(tc: List[List[bool]]) -\TEXTsymbol{>} None:
n = len(tc)
for k in range(n):
for i in range(n):
if tc[i][k]:
for j in range(n):
if tc[k][j]:
tc[i][j] = True
def generate\_posets(n: int):
pairs = [(i, j) for i in range(n) for j in range(i + 1, n)]
for mask in range(1 \TEXTsymbol{<}\TEXTsymbol{<} len(pairs)):
\# initialize the adjacency matrix for the identity relation (\TEXTsymbol{<}%
= reflexive)
tc = [[False] * n for \_ in range(n)]
for i in range(n):
tc[i][i] = True
\# apply subset of pairs after bitmask
for bit, (i, j) in enumerate(pairs):
if mask \& (1 \TEXTsymbol{<}\TEXTsymbol{<} bit):
tc[i][j] = True
floyd(tc) \# transitive closure
\# check antisymmetry
if any(i \TEXTsymbol{<} j and tc[i][j] and tc[j][i] for i in range(n) for j
in range(i+1, n)):
continue
\# force 0 bottom and 1 top
if not all(tc[0][x] for x in range(n)) or not all(tc[x][n-1] for x in
range(n)):
continue
yield tc
```

The result of this step is the set of all possible posets on n elements (labeled) with 0 minimum and 1 maximum. The complexity increases rapidly with n, because the number of labeled posets is very large (roughly, there are $2^{\binom{n}{2}}$ possible binary relations, although most will be eliminated by the antisymmetry conditions and 0, 1 extremes).

2. Lattice test and determination of $\wedge$ and $\vee$ operations. For each generated poset, it is checked whether it forms a lattice with 0 and 1. The lattice condition assumes the existence of an infimum ($\wedge$, meet) and a supremum ($\vee$, join) for any pair of elements in L. Basically, for each two elements $a, b \in L$, the set ($\{c \mid a \leq c \text{ and } b \leq c\}$) and the set ($\{d \mid d \leq a \text{ and } d \leq b\}$) are calculated. If either of these two sets does not have a unique minimum (respectively maximum) element, the structure is not a network and is rejected. Otherwise, the element $a \wedge b = glb\{a, b\}$ (the maximum of the first set) and $a \vee b = lub\{a, b\}$ (the minimum of the second set) are determined, building the tables of the binary operations $\wedge$ and $\vee$ on L. The following pseudocode extracts the $\wedge$ and $\vee$ operations by traversing all pairs of elements:

 doi:10.20944/preprints202510.0145.v1


```
def compute\_lattice(tc: List[List[bool]]):
n = len(tc)
meet = [[-1]*n for \_ in range(n)]
join = [[-1]*n for \_ in range(n)]
for a in range(n):
for b in range(a, n): \# we only consider a \TEXTsymbol{<}= b to avoid
duplication of pairs
\# the set of elements \TEXTsymbol{>}= a and \TEXTsymbol{>}= b (candidates
for a v b)
upper = \{c for c in range(n) if tc[a][c] and tc[b][c]\}
\# the set of elements \TEXTsymbol{<}= a and \TEXTsymbol{<}= b (candidates
for a \symbol{94}b)
lower = \{c for c in range(n) if tc[c][a] and tc[c][b]\}
lub = \{c for c in upper if not any(tc[c][d] and c != d for d in upper)\} \#
minimums in upper
glb = \{d for d in lower if not any(tc[d][c] and d != c for c in lower)\} \#
maximums in lower
if len(lub) != 1 or len(glb) != 1:
return None \# not a network (lub or glb is not unique)
j = lub.pop(); m = glb.pop()
join[a][b] = join[b][a] = j
meet[a][b] = meet[b][a] = m
return meet, join
```

If the above function returns `None` it means that the poset was not a valid network and is ignored. For accepted posets, we retain the $\wedge$ and $\vee$ tables for the next steps.

3. Enumeration of the $\otimes$ operation table (commutative monoid). Having fixed the lattice structure $((L, \wedge, \vee))$ for a valid poset, the next step is to generate all possible $\otimes$ operations (multiplications) that can complete the structure into a commutative monoid with the unit 1 (neutral element) and absorbing element 0. In addition, since we work with residual algebras (MTL/BL logics), we also impose the natural constraint $\forall a, b: \ a \otimes b \leq a \wedge b$ (the multiplication is at most as small as $\wedge$, element by element). The algorithm backtracks through all possible completions of the $\otimes$ table, cell by cell, respecting the following conditions during generation:

- **Initial conditions:** 0 is absorbing ($0 \otimes x = x \otimes 0 = 0$ for any x) and 1 is neutral ($1 \otimes x = x \otimes 1 = x$). These values are pre-filled in the $\otimes$ table before backtracking.

- **Commutativity:** The table is symmetric ($a \otimes b = b \otimes a$), so the algorithm only goes through half of the table cells (where $i \leq j$ for the row vs. column index).

- **Constraint $\otimes \leq \wedge$:** For any a, b, the value assigned to $a \otimes b$ cannot exceed (in the sense of the $\leq$ order of the lattice) $a \wedge b$. In numerical terms (given that the elements are $0, \ldots, n-1$ in the $\leq$ order), this means $a \otimes b \leq a \wedge b$; so if $m = a \wedge b$ (from the previously computed meet table), then $a \otimes b$ must be chosen from the set $\{0, 1, \ldots, m\}$.

We recursively explore each empty cell of the $\otimes$ table, assigning it in turn an admissible value (respecting commutativity and $\otimes \leq \wedge$) and rejecting (*backtrack*) if any condition is violated. Finally, when all cells are filled, the *associativity* of $\otimes$ is checked on the entire table (the condition $(a \otimes b) \otimes c = a \otimes (b \otimes c)$ for all a, b, c). Only associative tables are kept as valid solutions. Below is a code fragment illustrating the basic $\otimes$ table generation scheme (without additional optimizations):

 doi:10.20944/preprints202510.0145.v1


```
def associative(tbl: List[List[int]]) -\TEXTsymbol{>} bool:
n = len(tbl)
return all(tbl[tbl[a][b]][c] == tbl[a][tbl[b][c]]
for a in range(n) for b in range(n) for c in range(n))
def gen\_monoid(meet: List[List[int]],tc:List[List[bool]], n: int):
top = n - 1
\# initialize the * table with -1 (unknown) and set the initial conditions
base = [[-1] * n for \_ in range(n)]
for i in range(n):
base[i][0] = base[0][i] = 0 \# 0 * i = 0
base[i][top] = base[top][i] = i \# 1 * i = i
cells = [(i, j) for i in range(n) for j in range(i, n) if base[i][j] == -1]
def backtrack(k: int):
if k == len(cells): \# basis of recursion: full table
tbl = [[base[i][j] if base[i][j] != -1 else base[j][i]
for j in range(n)] for i in range(n)]
if associative(tbl):
yield tbl \# produces a valid solution
return
i, j = cells[k]
\# iterate through possible values 0..meet[i][j]
        m = meet[i][j]
        candidates = [d for d in range(n) if tc[d][m]]
for val in candidates:
base[i][j] = base[j][i] = val
yield from backtrack(k + 1)
base[i][j] = base[j][i] = -1 \# backtrack: reset
yield from backtrack(0)
```

In the above code, `meet` is the computed table of $\wedge$ (which gives the upper bound for $\otimes$), and the list `cells` contains the coordinates in the upper triangle of the table (including the diagonal) that need to be filled. At the time of initial generation (without optimizations), backtracking tries all possible values between the default lower bound 0 and the upper bound `meet[i][j]` for each cell, which leads to a combinatorial explosion as n increases. The complexity of this step is, in the worst case, exponential in the number of unassigned cells (approximately $\frac{n(n+1)}{2} - (2n-1)$ after setting the initial conditions). However, for small n (e.g. $n \leq 6$), this generation is feasible and produces all candidate tableaux $\otimes$ compatible with the given network.

4. Calculating residuated $\rightarrow$ and checking the adjunction property. For each generated structure $(L, \wedge, \vee, \otimes)$, we compute the implication operation $\rightarrow$ as the residue of $\otimes$. Specifically, for any two elements $a, b \in L$, we define:

$$a \rightarrow b \;=\; \max\{\, d \in L \mid a \otimes d \leq b \,\},$$

where the order $\leq$ is that of the network (determined by the poset). The intuition is that $a \rightarrow b$ is the largest element d that, multiplied by a, gives a result $\leq b$. If this maximum is not unique (or does not exist), then the current structure does not satisfy the axiom and is rejected. After computing the tableau of $\rightarrow$, the adjunction is checked: for all $a, b, c \in L$ we must have $a \otimes b \leq c$ if and only if $b \leq a \rightarrow c$. If there is any triple (a, b, c) that violates this equivalence, the structure is invalidated. In practice, once $\otimes$ is associative and $\rightarrow$ is defined as above, the adjunction property is guaranteed by construction, but the implementation explicitly checks this as a safety measure.

 doi:10.20944/preprints202510.0145.v1

5. Classifying the result and eliminating isomorphic doubles. At this point, if a structure $(L, \wedge, \vee, \otimes, \rightarrow, 0, 1)$ has passed all the axiomatic tests, it is classified based on the prelinearity and divisibility properties. This is done by checking the definition conditions mentioned above: if $(\forall x, y,\ (x \rightarrow y) \vee (y \rightarrow x) = 1)$ is true in the structure, then prelinearity is satisfied; similarly for divisibility $(\forall x, y,\ x \otimes (x \rightarrow y) = x \wedge y)$. Based on these two properties, each generated model is labeled as pure MTL (prel=true, div=false), pure DIV (div=true, prel=false) or BL (prel=true, div=true).

Finally, to avoid redundant enumerations, an **isomorphic deduplication** is applied: many generated structures can be equivalent under element renaming (*algebra isomorphism*). Initially, the algorithm treats all elements in L as distinctly labeled, so the same algebras can appear several times under different permutations of the labels $\{0, \ldots, n-1\}$. To filter out isomorphic duplicates, we proceed as follows: we consider all σ permutations of the set $\{0, \ldots, n-1\}$ that fix elements 0 and 1 (since we conventionally choose to represent each isomorphic class by having 0 and 1 in their standard roles). For each such permutation, σ is applied to the structure labels (on all of them: on the order relation, on the implication and on the operation tables) and the result is serialized (for example, one can concatenate the elements of the matrix of $\otimes$ and the upper triangle of the matrix $\leq$ together with the matrix of $\rightarrow$ into a string). The canonical representative of the isomorphism class is chosen as the minimal lexicographical string obtained from all possible permutations. The algorithm stores only the structures whose serialized string is minimal â€“ eliminating those that do not correspond to the canonical representative (they are duplicates). This method ensures that each isomorphic algebra class is reported only once.

In the initial implementation, isomorphic deduplication has a significant cost, since the number of permutations to check is $(n-2)!$ (all permutations of interior elements, except 0 and 1). For n up to 6 or 7 this is feasible, but quickly becomes impractical for larger n.

Basic complexity summary: The search space of the basic algorithm is dominated by the number of labeled posets (step 1) and the degrees of freedom in filling the $\otimes$ table (step 3). Even with the constraint $\otimes \leq \wedge$, the number of possible monoidal tableaus remains huge, causing the running time to grow exponentially with n. In practical terms, the first version of the code was able to generate the complete algebras up to order $n = 6$ in reasonable time, but at $n = 7$ the running time exploded exponentially, exceeding acceptable limits. This observation motivated the development of targeted optimizations, described in the following section.

6. Currently Implemented Optimizations

To extend the exploration to higher-order algebras and reduce the computational time, several optimizations have been added to the algorithm. Below we present each such optimization, explaining the changes made compared to the initial version and the practical effect (reduction of execution time by *pruning* of the search space).

7. Optimized Filling Order of the $\otimes$ Table

In the initial version, the cells of the $\otimes$ operation table were filled in an implicit order (i.e., by traversing the rows and columns in lexicographic sequence). The current version modifies this strategy by *sorting the list of unfilled cells* in increasing order of the values of the infimum ($\wedge$) for each pair of elements. In other words, the cells (a, b) are filled according to the constraint $\otimes \leq \wedge$, so that pairs with smaller $a \wedge b$ (and therefore potentially more restrictive as admissible values) are explored first. Compared to the old version, which treated the cells in a fixed order, this sorting causes conflicts to appear earlier in the backtracking process, eliminating impossible branches at a lower depth of the search tree. Basically, by filling the positions with fewer possible options first, early *pruning* is achieved and the exploration is considerably accelerated.

 doi:10.20944/preprints202510.0145.v1

8. Incremental Associativity Checks

The current implementation also performs *partial associativity checks* during the construction of the $\otimes$ table. More precisely, after each new assignment $a \otimes b = v$, the algorithm tests whether this value immediately produces a violation of the associative relation $(x \otimes y) \otimes z = x \otimes (y \otimes z)$ for a triplet of elements x, y, z in L whose partial products have already been established. The check is *incremental*: only the cases "visible" in the current state of the table are considered (situations in which expressions of the form $(x \otimes y) \otimes z$ and $x \otimes (y \otimes z)$ can be completely evaluated based on the already known values). If any such situation is identified in which the associativity equality is violated by the current value v, then the respective completion cannot lead to a final associative table and the branch is immediately abandoned.

9. Differences and Practical Impact

In the original version, the associativity of $\otimes$ was checked *only at the end*, after the entire table was complete. This meant that many branches were explored in full, although they may have contained a latent associative inconsistency from the middle. The new version, by partially checking associativity, manages to detect some of these incompatibilities much earlier. Although the associativity test is expensive in general, its local form applied at each step is feasible and brings pruning benefits: it reduces the number of unnecessary additions and contributes to the overall efficiency of backtracking.

10. Representing Posets with Bitmasks

Each element of the poset and its successors can be encoded as a bitmask, where the bit k indicates whether the current element is $\leq$ to the element k. This representation transforms set operations (e.g. union or intersection of successors) into very fast bitwise operations: intersection becomes a logical AND, union becomes a logical OR, etc. Thus, transitive closure or composition calculations are reduced to bitwise operations, significantly speeding up the algorithm.

11. Incremental Propagation of Transitive Relations

In the incremental construction of the poset, each time a new comparability relation $(i \leq j)$ is added, the transitive closure is immediately updated. Basically, when j becomes a successor of i, we update the bit mask of i with the OR of the mask of j, thus capturing all its transitive descendants. This operation is equivalent to one step in the Warshall algorithm (update with OR/AND), but is done incrementally, thus avoiding complete recalculations of the closure.

12. Optimizing the Residuation Calculation with Bit Mask

The internal residuation $a \to b$ (the implication in the poset) is calculated by finding the maximal element x such that $a * x \leq b$. Using the bit representation, the candidate set is filtered by bitwise operations: we intersect the bit masks of the possible successors of a with the condition $x \leq b$. Since bitwise operations are very fast (significantly faster than complex arithmetic operations), this procedure considerably speeds up the residue determination.

By combining the above optimizations (cell sorting, associativity checks during generation, use of bitwise operations), the current version of the algorithm manages to explore the space of possibilities much more efficiently. Experimentally, these improvements extended the threshold of complete enumeration up to $n = 7$ (all algebras of order 7 could be generated at a reasonable cost), which was not feasible in the original version (which basically stopped at $n = 6$, before becoming unaffordable). However, for $n \geq 8$, the search space remains prohibitive even with the implemented optimizations, which motivates the development of additional strategies, described in the next section.

13. Planned Future Optimizations

Although the current optimizations have pushed the model enumeration frontier up to $n = 7$, there are still significant limitations that prevent the algorithm from scaling to larger n. The main

 doi:10.20944/preprints202510.0145.v1

obstacles are: (a) the fact that the generation of networks (posets) is still performed at the level of labeled structures (even if many duplicates are subsequently filtered out, the complexity of poset enumeration increases greatly for large n), and (b) the cost of isomorphism checking based on generating all permutations increases exponentially with increasing n.

To overcome these limitations and extend the exploration to higher orders ($n = 8, 9, 10$ and above), several optimizations and additional approaches are planned, as follows.

14. Canonization at the Level of the Order Relation (Poset)

Instead of the algorithm enumerating and testing *all* labeled posets (step 1) to discover the isomorphically distinct ones, we plan to apply a *canonization* of the poset itself during generation. The idea is to define a canonical criterion (e.g., lexicographic) for representing the transitivity matrix of the relation $\leq$ under the action of permutations of the set L (fixing the elements 0 and 1). Thus, when generating the posets, we will be able to skip those that are isomorphic to already generated ones, keeping only the canonical representative of each order isomorphism class.

The planned concrete implementation would group the intermediate elements $2, 3, \ldots, n-2$ into equivalence classes based on simple invariants (such as the number of elements above and below in the poset, i.e. the values $|down(x)|$ and $|up(x)|$ for each element x). Then, by backtracking, only permutations that respect these groupings would be explored (i.e. only elements with the same $(|down|, |up|)$ are permuted among themselves). For each generated poset, the minimal representative string (serialization of the upper triangle of the adjacency matrix $\leq$) will be determined and used as the canonical identifier. If a newly generated poset has the same identifier as one already seen, it will be ignored.

Expected benefit: Such canonicalization would eliminate duplicate isomorphic networks from the very beginning, dramatically reducing the number of posets on which we have to run the subsequent steps (completion of $\otimes$ etc.). Basically, the effort of exploring posets would decrease from enumerating all labeled ones (which contain redundancy of order $n!$ in the worst case) to enumerating only the non-isomorphic ones. For example, for $n = 7$, this optimization would bring a substantial gain, since the number of distinct networks is much smaller than the number of raw labeled networks.

15. Dynamic Lower Bounds and Local Prunes in Backtracking for $\otimes$

The most difficult step remains the filling of the table for the $\otimes$ operation. To make backtracking more efficient and avoid exploring obviously impossible configurations, two types of optimizations will be added to the generation of $\otimes$:

- **Calculating a dynamic lower bound:** In addition to the already used upper bound $ub(i, j) = a \wedge b$ (i.e. $a \otimes b \leq a \wedge b$), we will also introduce a *lower bound* $lb(i, j)$ dynamically calculated for each cell (i, j) being filled. Intuitively, $lb(i, j)$ represents the smallest value that $a = i \otimes j$ would be constrained to take, given the values already established in the table. It will be calculated as follows:

$$lb(i, j) = \bigvee \{\, c \otimes d \mid c \leq i,\ d \leq j \text{ and } c \otimes d \text{ is already established} \,\},$$

i.e. the join of all known products $c \otimes d$ where $c \leq i$ and $d \leq j$. This set includes, in particular, the cases $c = i$ or $d = j$ if these products have already been filled in. The lower bound $lb(i, j)$ will be calculated incrementally as the table is filled in, using the current values in the table and the table of $\vee$. **Anticipated impact:** Instead of backtracking the value $a \otimes b$ from 0 to $a \wedge b$, we will be able to start directly from $lb(a, b)$, considerably reducing the number of candidate values tried for that cell.

- **Local consistency checks (prunes):** After assigning a provisional value to a cell $a \otimes b = v$, two local tests will be applied immediately to detect violations of the monotonicity and right distributivity properties:

- *Partial monotonicity:* If $x \leq y$ in the poset, then for any known z we must have $x \otimes z \leq y \otimes z$ and $z \otimes x \leq z \otimes y$. During the completion of the table, not all values are determined, but if both $x \otimes z$ and $y \otimes z$ (or symmetrically $z \otimes x$ and $z \otimes y$) have already been determined, we will immediately

 doi:10.20944/preprints202510.0145.v1

check that the order relationship between the results corresponds to that between the operands. Any inversion (i.e. we find $x \leq y$, but the value assigned to $x \otimes z$ is not $\leq$ than $y \otimes z$) indicates a violation of monotonicity, and the branch will be pruned (*pruned*) on the spot.

- Right distributivity (local):

The right distributivity axiom in residual networks states that $a \otimes (b \vee c) = (a \otimes b) \vee (a \otimes c)$ for any a, b, c. During partial generation, we will not be able to verify this formula completely for all a, b, c (many values are missing), but we will identify local cases where both $a \otimes b$, $a \otimes c$, and $a \otimes (b \vee c)$ have been established. If in such restricted situations the relation $a \otimes (b \vee c) = (a \otimes b) \vee (a \otimes c)$ is violated by the current assignment, then we will know for sure that the full extension will not satisfy the right distributivity, so the branch can be stopped early.

It is anticipated that, with these improvements, the backtracking procedure for monoids will become much more efficient. The code snippet below shows a possible pseudocode for an optimized generation of $\otimes$, integrating the calculation of bounds and local prunes into the recursive loop:

```
def lower\_bound\_for\_cell(base, i, j, tc, join) -\TEXTsymbol{>} int:
"""Calculate lb(i,j) based on the values already established in the base
table."""
n = len(base); current\_lb = None
for c in range(n):
if not tc[c][i]: \# c \TEXTsymbol{<}= i ?
continue
for d in range(n):
if not tc[d][j]: \# d \TEXTsymbol{<}= j ?
continue
v = base[c][d]
if v \TEXTsymbol{<} 0:
continue \# if c*d is not yet set, skip it
current\_lb = v if current\_lb is None else join[current\_lb][v]
return current\_lb if current\_lb is not None else 0
def partial\_monotone\_ok(base, tc, i, j) -\TEXTsymbol{>} bool:
"""Check partial monotonicity on known rows/columns i,j."""
n = len(base); vij = base[i][j]
for d in range(n):
vd = base[i][d] \# i * d
if vd \TEXTsymbol{>}= 0: \# if i*d is known
\# i \TEXTsymbol{<}= j =\TEXTsymbol{>} (i*d) \TEXTsymbol{<}= (j*d) ?
if tc[j][d] and not tc[vd][vij]:
return False
\# j \TEXTsymbol{<}= d =\TEXTsymbol{>} (i*d) \TEXTsymbol{>}= (j*d) ?
if tc[d][j] and not tc[vij][vd]:
return False
for c in range(n):
vc = base[c][j] \# c * j
if vc \TEXTsymbol{>}= 0:
\# c \TEXTsymbol{<}= i =\TEXTsymbol{>} (c*j) \TEXTsymbol{<}= (c*i) ?
if tc[i][c] and not tc[vc][vij]:
return False
\# i \TEXTsymbol{<}= c =\TEXTsymbol{>} (c*j) \TEXTsymbol{>}= (c*i) ?
if tc[c][i] and not tc[vij][vc]:
```

 doi:10.20944/preprints202510.0145.v1


```
return False
return True
def partial\_rdist\_ok(base, join, i, j) -\TEXTsymbol{>} bool:
"""Check the local right distributivity for row i and column j."""
n = len(base)
for b in range(n):
for c in range(n):
if join[b][c] == j: \# if (b V c) = j in lattice
ab = base[i][b] \# a*b (a = i)
ac = base[i][c] \# a*c
if ab \TEXTsymbol{>}= 0 and ac \TEXTsymbol{>}= 0:
\# compare a*(b V c) with (a*b) V (a*c)
if base[i][j] != join[ab][ac]:
return False
return True
def gen\_monoid\_with\_bounds(meet, join, tc, n):
top = n-1
base = [[-1]*n for \_ in range(n)]
\# initializations 0 absorbent and 1 neutral as before
for i in range(n):
base[i][0] = base[0][i] = 0
base[i][top] = base[top][i] = i
cells = [(i, j) for i in range(n) for j in range(i, n) if base[i][j] == -1]
\# sort the cells so that the most restrictive ones (with smaller meet) are
filled first
cells.sort(key=lambda ij: meet[ij[0]][ij[1]])
def backtrack(k: int):
if k == len(cells):
\# complete table, will check associativity at the end
tbl = [[base[i][j] if base[i][j] != -1 else base[j][i]
for j in range(n)] for i in range(n)]
if associative(tbl):
yield tbl
return
i, j = cells[k]
\# dynamically calculate the bounds for this cell
lb = lower\_bound\_for\_cell(base, i, j, tc, join)
ub = meet[i][j]
        candidates = [d for d in range(n) if tc[d][ub] and tc[lb][d]]
for val in candidates:
base[i][j] = base[j][i] = val
\# apply local prunes before recursively descending
if partial\_monotone\_ok(base, tc, i, j) and partial\_rdist\_ok(base, join,
i, j):
yield from backtrack(k + 1)
base[i][j] = base[j][i] = -1 \# return (backtrack)
yield from backtrack(0)
```

 doi:10.20944/preprints202510.0145.v1

In the above pseudocode, the function `gen_monoid_with_bounds` represents the planned optimized version of the monoid generator. The main observations about this approach would be:

- The list of cells to be filled will be sorted in ascending order of $\wedge$ values (cells with small meet, hence potentially more restrictive, will be approached first to cause early prunes).

- For each cell (i, j), the values tried would be from the new lower bound `lb` (computed based on the values already set) up to `ub` = `meet[i][j]`. Thus, a lot of unnecessary lower values would be automatically skipped that would have been implicitly eliminated by the order constraints anyway.

- After setting a candidate value `val` in the cell, `partial_monotone_ok` and `partial_rdist_ok` will be checked immediately. These functions test local monotonicity and distributivity conditions based on the values already known in the table (note that they only use `base[...]` entries that have been filled in, the unfilled ones being -1 and not influencing the checks). If any test fails, the branch will be cut without continuing deeper.

- *Full associativity* will still be checked only at the end, at the backtracking leaf (it is more efficient to leave the full global associativity check at the end, although theoretically it could also be tested incrementally in part).

Expected benefits of these optimizations: The introduction of dynamic bounds and local prunes would drastically reduce the number of explored $\otimes$ table configurations. Branches that would lead to monotonicity or distributivity violations would be eliminated very early, and the ranges of possible values for each cell would be significantly narrowed. Based on the performances observed in current optimizations, we estimate that this optimized version of backtracking could lead to a jump from the impossibility of completing the case $n = 8$ (with the original algorithm) to obtaining the full result for $n = 8$ in reasonable time. However, for $n = 9$, even with these improvements, the search space would still be too large, which will probably require further optimizations or a change in strategy (see the discussion below for other directions).

16. Isomorphic Deduplication Based on the Network Automorphism Group

The last major planned optimization aims to reduce the cost of isomorphic deduplication (the final step of the algorithm). Instead of testing all $(n-2)!$ permutations for each structure (to determine the canonical representative of the isomorphism class), we plan to explicitly compute the *automorphism group* $\mathrm{Aut}(L)$ of the current network $(L, \leq)$ and use only this group (a much smaller set of permutations) to identify the isomorphism classes of the $\otimes$ tables.

An *automorphism* of the network $(L, \leq)$ is a σ permutation of the set L that preserves the order relation (i.e. $\forall x, y : \; x \leq y \iff \sigma(x) \leq \sigma(y)$). The automorphisms form a group (under composition), and the size of this group depends on the internal symmetries of the poset. For example, if the network is a completely ordered chain, then $\mathrm{Aut}(L)$ will be practically trivial (just the identity), while if the network has non-trivial symmetries (e.g. two incomparable interchangeable elements), the group will contain more non-trivial permutations.

The optimized algorithm would determine $\mathrm{Aut}(L)$ by a backtracking similar to poset canonization: we will group equivalent elements (according to invariants of degree, level, etc. in the poset) and try permutations of these elements, checking the invariance of the relation $\leq$. This will obtain the list of all poset automorphisms. Then, for a given structure (a certain completion $\otimes$ on the respective poset), the canonical representative of the isomorphism class can be computed without exploring all possible permutations, but by applying only each automorphism $\sigma \in \mathrm{Aut}(L)$ to the table of $\otimes$ and choosing the encoding (serialized string) that is lexicographically smallest. Intuitively, we will restrict ourselves to the relevant permutations (poset symmetries); permutations that are not automorphisms of the network will not produce structurally distinct algebras, since they move elements between positions that are not structurally equivalent.

Estimated benefit: This improvement would drastically reduce the complexity factor of deduplication. Instead of $(n-2)!$ permutations (which for $n = 7$ means $5! = 120$ cases, and for $n = 8$ would be $6! = 720$ and so on), only $|\mathrm{Aut}(L)|$ permutations would be analyzed. Usually, $|\mathrm{Aut}(L)|$ is much

 doi:10.20944/preprints202510.0145.v1

smaller than the factorial, especially for "asymmetric" posets. In ideal cases, the poset has very few automorphisms (only the identity), so practically no additional comparison is needed; even in cases with non-trivial symmetries, there would be enormous savings compared to the naive approach. With this optimization, we expect that the memory and time consumption for isomorphism filtering would decrease considerably, especially at the upper bounds of the explored n.

In addition to the major optimizations described above, we explore other possible directions for improving the algorithm, such as:

- **Direct generation of *non-isomorphic* posets:** Instead of enumerating all labeled posets and canonizing them later, one can adopt a canonical augmentation strategy (used for example in graph or non-isomorphic poset generation algorithms). This would build the networks gradually, adding one element at a time and imposing canonical conditions at each step, so that each non-isomorphic network is generated only once. Such a method would completely eliminate isomorphic redundancy already in the poset generation phase, saving enormously for large n.

- **Using a dedicated solver for automorphisms:** Automorphism finding problems can be approached much more efficiently with graph theory tools,
such as `nauty`-type algorithms or other specialized solvers. By transforming the network (or network with operations) into a graph and using such tools, we could quickly compute the automorphism group even for larger structures, avoiding the manual backtracking implemented in Python.

- **Other heuristic optimizations and parallel execution:** For example, implementing additional *symmetry breaking* in the generation of $\otimes$ (such as conditions that fix certain order relations between the rows in the $\otimes$ table to avoid equivalent permutations) or running the algorithm in parallel (distributing the input posets between multiple computational processes) could bring significant performance gains. Also, any additional theoretical knowledge about these algebras (new logical constraints, structural theorems) could be used to reduce the search space.

In conclusion, the presented algorithm, together with the current optimizations, represents an efficient compromise between generality and efficiency in the problem of generating finite residual algebras. It allowed the complete exploration of the model space for small sizes $n = 7$ and provides a platform from which it can be extended to even higher orders by developing the optimization ideas discussed above.

In the specialized literature, such algebras up to $n = 12$ are known, through an alternative algorithm. By the present different approach, we succeeded in generating algebras up to order $n = 7$. By the proposed optimizations we believe that the order $n = 12$ can be exceeded and high orders can be reached.

Regarding the number of algebras in each analyzed family, we obtained the following results after the programs have been run (see the following table):

ine Order	pure MTL	pure DIV	BL	other RL	ALL
ine 1	0	0	1	0	1
ine 2	0	0	1	0	1
ine 3	0	0	2	0	2
ine 4	2	0	5	0	7
ine 5	14	1	9	2	26
ine 6	79	3	20	27	129
ine 7	426	11	38	248	723
ine					

By „pure MTL" we understand MTL-algebras which are not BL-algebras and by „pure DIV" we understand DIV algebras which are not BL-algebras, by „other RL" we understand residuated lattices which are not MTL, DIV or BL and by „ALL", we understand all residuated lattices.

Conclusions. In this paper we present some properties of divisible residuated lattices and MTL-algebras. These structures have a important significance in the study of fuzzy logic. From computational considerations, we analyzed the structure of these residuated lattices of small size n

 doi:10.20944/preprints202510.0145.v1

($2 \leq n \leq 5$) and we gave summarizing statistics. These results can be extended for higer size, by using computer and a constructive algorithm for generating all residuated lattices.

References

[BV; 10], Belohlavek, R., Vychodil, V., *Residuated Lattices of size*$\leq$ 12, Order, 27(2010), 147-161.

[Bl; 53] Blair, R.L, *Ideal lattices and the structure of rings*, Trans. Am. Math. Soc., 75(1953), 136–153.

[BP; 02] Busneag, D., Piciu, D., *Lectii de algebra*, Ed. Universitaria, Craiova, **2002**.

[CFP; 23] Călin, M. F., Flaut, C., Piciu, D., Remarks regarding some Algebras of Logic, Journal of Intelligent & Fuzzy Systems, 45(5)(2023), Journal of Intelligent & Fuzzy Systems, 45(5)(2023), 8613-8622, DOI: 10.3233/JIFS-232815.

[COM; 00] Cignoli, R.; D'Ottaviano, I.M.L.; Mundici, D., *Algebraic Foundations of many-valued Reasoning*. Trends in Logic-Studia Logica Library 7, Dordrecht: Kluwer Acad. Publ.**2000**.

[CHA; 58] Chang, C.C., *Algebraic analysis of many-valued logic*, Trans. Amer. Math. Soc. 88(1958), 467-490.

[NL; 03] Di Nola, A., Lettieri, A., *Finite BL-algebras*, Discrete Mathematics, 269 (2003), 93—112.

[Di; 38] Dilworth, R.P., *Abstract residuation over lattices*, Bull. Am. Math. Soc. 44(1938), 262–268.

[E] Esteva, F., Godo, L., *Monoidal t-norm based logic : towards a logic for left-continuous t-norms*, Fuzzy Sets and Systems, **124** (2001), 271-288.

[F] Fodor, J.K., *Contrapositive symmetry of fuzzy implications*, Fuzzy Sets and Systems, **69** (1995), 141-156.

[FP; 22] Flaut, C., Piciu, D., *Connections between commutative rings and some algebras of logic*, Iranian Journal of Fuzzy Systems, **19** (6) (2022), 93-110.

[FP; 23] Flaut, C., Piciu, D.,*Commutative Rings Behind Divisible Residuated Lattices*, Mathematics 2024, 12(23), 3867; https://doi.org/10.3390/math12233867

[I; 09] Iorgulescu, A., *Algebras of logic as BCK algebras,* A.S.E., Bucharest, **2009**.

[KS; 24] Koohnavard R., Saeid A.B., Residuated skew lattices with modal operator, An. Şt. Univ. Ovidius Constanţa, 31(1),2023, 167–179, DOI: 10.2478/auom-2023-0008.

[S] Swamy K.L.N., *Dually residuated lattice ordered semigroups*, Mathematische Annalen, 159 (2) (1965), 105-114.

[TT; 22] Tchoffo Foka, S. V., Tonga, M., *Rings and residuated lattices whose fuzzy ideals form a Boolean algebra*, Soft Computing, 26 (2022) 535-539.

[T; 99] Turunen, E., *Mathematics Behind Fuzzy Logic*, Physica-Verlag, **1999**.

[WD; 39] Ward, M., Dilworth, R.P., *Residuated lattices*, Trans. Am. Math. Soc. 45(1939), 335–354.