Residuated Lattices of Size ≤ 12
extended version*

Radim Belohlavek[1,2], Vilem Vychodil[1,2]

[1] Dept. Computer Science, Palacky University, Olomouc
17. listopadu 12, Olomouc, CZ–771 46, Czech Republic

[2] SUNY Binghamton, PO Box 6000, Binghamton, NY 13902–6000, USA
email: rbelohla@binghamton.edu, vychodil@binghamton.edu

Abstract

We present the numbers of all non-isomorphic residuated lattices with up to 12 elements and a link to a database of these lattices. In addition, we explore various characteristics of these lattices such as the width, length, and various properties considered in the literature and provide the corresponding statistics. We also present algorithms for computing finite lattices and finite residuated lattices including a fast heuristic test of non-isomorphism of finite lattices.

1 Introduction

Computing non-isomorphic finite structures of a particular type with up to a given, possibly large, number of elements is important for two reasons. First, one gets the numbers of non-isomorphic structures up to a given size. This if beneficial if direct formulas for these numbers, or their estimates, are not available. Second, one gets a database of non-isomorphic structures up to a given size. Such database is useful when looking for examples or counterexamples of the particular structures. A further exploration of the examples may lead to various conjectures regarding the structures and, in general, may provide us with further insight regarding these structures.

In this paper, we explore finite residuated lattices with up to 12 elements. In particular, we present the numbers of all non-isomorphic lattices and residuated lattices with up to 12 elements and provide a link to a database where one can find all these lattices. We also present algorithms for computing finite lattices and finite residuated lattices which we used for computing the database. In addition, we explore various properties and characteristics of these lattices and provide the corresponding summaries.

Ordered sets and lattices play a crucial role in several areas of mathematics and their applications, e.g. in data visualization and analysis, uncertainty modeling, many-valued and fuzzy logics, graph theory, etc. Residuated lattices, in particular, were introduced in the 1930s by Dilworth and Ward [10, 35]. In the late 1960s, residuated lattices were introduced into many-valued logics and, in particular, into fuzzy logics as structures of truth values (truth degrees) [16, 17]. Residuated lattices and various special residuated lattices are now used as the main structures of truth values in fuzzy logic and fuzzy set theory, see e.g. [4, 12, 18, 20, 23, 29, 33], and are subject to algebraic investigation, see e.g. [5, 15, 25].

In addition to the above-mentioned general motivations for exploring finite residuated lattices, there is one more. Namely, the role of residuated lattices in fuzzy set theory is that they serve as scales of truth degrees. In many application areas, see e.g. [29], the scenario is the following. An expert defines a fuzzy set by assigning truth degrees (elements of a residuated lattice) to the elements of a particular universe. Now, according to Miller's 7 ± 2 phenomenon, well-known from psychology [32], humans are able to assign degrees in a consistent manner provided the scale of degrees contains up to 7 ± 2 elements. With more than 7 ± 2 elements, the assignments become inconsistent. From this perspective, by computing all residuated lattices with up to 12 elements, we cover all the residuated lattices which are practically useful in such scenario, i.e. those with up to 7 ± 2 elements.

*This version of the paper contains full proofs and detailed descriptions of algorithms including pseudo-codes. The original short version of the paper will appear in *Order*, see `http://dx.doi.org/10.1007/s11083-010-9143-7`.

A previous work on related problems includes [24] where the author provides the numbers and descriptions of non-isomorphic residuated lattices with up to 6 elements. The results were computed using the GAP system for computational discrete algebra. [2, 8] are also studies related to our paper. Namely, the authors compute the numbers of all t-norms [28] on finite chains and do not pay attention to general nonlinear residuated lattices. [2] contains a summary of numbers of t-norms and their basic properties for chains with $n \leq 11$ elements. A similar approach is used in [8, 9] which shows the numbers of t-norms for chains with $n \leq 14$ elements. In [22], the numbers of all finite lattices with up to 18 elements are presented along with the algorithm. With respect to the previous work, we improve the size up to which we compute all the residuated lattices, from 6 (see [24]) to 12. Moreover, we systematically explore various properties of the residuated lattices which has not been provided in the previously published papers. Since the algorithmic aspects of generating residuated structures are scarcely discussed in the literature, we present a detailed description of algorithms we have used to generate the structures of our interest.

The paper is organized as follows. Section 2 presents preliminaries and notation we use. Section 3 describes algorithms for generating finite lattices including heuristic tests of non-isomorphism. In Section 4 we present algorithms for generating finite residuated lattices. In Section 5 we present a summary regarding selected properties of the computed structures. Tables summarizing observed properties can be found in the appendix.

2 Preliminaries and Notation

In this section we introduce basic notions and notation. Details can be found e.g. in [3, 19] (ordered sets and lattices) and [4, 15, 18, 20, 23, 25] (residuated lattices).

Recall that a partial order in L is a binary relation on L which is reflexive, antisymmetric, and transitive. If $\leq$ is a partial order on L, the couple $\mathbf{L} = \langle L, \leq \rangle$ is called a partially ordered set. If there exists a least or a greatest element of $\langle L, \leq \rangle$, it is denoted by 0 or by 1, respectively. A lattice which has both 0 and 1 is called a bounded lattice. For each $A \subseteq L$ we denote by $\mathcal{L}(A)$ and $\mathcal{U}(A)$ the lower and upper cones of A, respectively. $\mathcal{L}(A)$ and $\mathcal{U}(A)$ are defined by

$$\mathcal{L}(A) = \{b \in L \,|\, b \leq a \text{ for each } a \in A\}, \tag{1}$$

$$\mathcal{U}(A) = \{b \in L \,|\, a \leq b \text{ for each } a \in A\}. \tag{2}$$

If $\mathcal{L}(A)$ has a greatest element a, then a is called the infimum of A in $\langle L, \leq \rangle$, denoted by $\bigwedge A$. Dually, if $\mathcal{U}(A)$ has a least element a, then a is called the supremum of A in $\langle L, \leq \rangle$, denoted by $\bigvee A$. A partially ordered set $\langle L, \leq \rangle$ is a lattice if infimum and supremum exist for any two-element subset of L. As usual, we write $a \wedge b$ and $a \vee b$ instead of then $\bigwedge\{a, b\}$ and $\bigvee\{a, b\}$. A partially ordered set $\langle L, \leq \rangle$ is called linearly ordered (or, a chain) if any two elements $a, b \in L$ are comparable, i.e. $a \leq b$ or $b \leq a$. Let $\mathbf{L}_1 = \langle L_1, \leq_1 \rangle$ and $\mathbf{L}_2 = \langle L_2, \leq_2 \rangle$ be lattices. A map $h \colon L_1 \to L_2$ is called a lattice isomorphism (between lattices $\mathbf{L}_1$ and $\mathbf{L}_2$) if (i) h is a bijection, and (ii) for each $a, b \in L_1$, we have

$$a \leq_1 b \quad \text{iff} \quad h(a) \leq_2 h(b). \tag{3}$$

Lattices $\mathbf{L}_1$ and $\mathbf{L}_2$ are *isomorphic,* written $\mathbf{L}_1 \cong \mathbf{L}_2$, if there exists a lattice isomorphism between $\mathbf{L}_1$ and $\mathbf{L}_2$. Throughout the paper, we consider lattices "up to isomorphism", i.e. we tacitly identify all isomorphic lattices. As usual, we freely interchange lattices considered as special partially ordered sets and lattices considered as algebras with two binary operations of meet and join. Hence, $\mathbf{L} = \langle L, \leq \rangle$ and $\mathbf{L} = \langle L, \wedge, \vee \rangle$ denote the same structure.

A residuated lattice is an algebra $\mathbf{L} = \langle L, \wedge, \vee, \otimes, \to, 0, 1 \rangle$ where $\langle L, \wedge, \vee, 0, 1 \rangle$ is a bounded lattice, $\langle L, \otimes, 1 \rangle$ is a commutative monoid, and $\otimes$ and $\to$ satisfy $a \otimes b \leq c$ iff $a \leq b \to c$ for each $a, b, c \in L$ (adjointness property). Binary operations $\otimes$ (multiplication) and $\to$ (residuum) serve as truth functions of connectives "fuzzy conjunction" and "fuzzy implication" [4, 12, 18, 17, 20, 23]. Various subclasses of residuated lattices have been investigated in many-valued and fuzzy logics, e.g. MTL-algebras [12], BL-algebras [20] and its three important subclasses, namely MV-algebras, Gödel algebras, and Π-algebras.

3 Generation of non-isomorphic finite lattices

In this section we present a method for computing all non-isomorphic finite lattices of a given size. The process of generation of finite lattices includes several issues. We need an algorithm that generates lattices one by one so that for all isomorphic lattices, the algorithm generates just one representative of them. In our approach, we

$\leq$	0	a	b	c	d	1
0	×	×	×	×	×	×
a		×			×	×
b			×	×	×	×
c				×		×
d					×	×
1						×

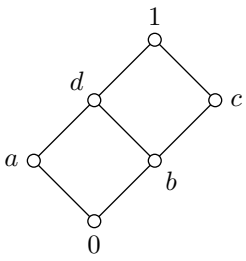


Figure 1: Upper triangular adjacency matrix (left) of a finite lattice (right).

propose a brute-force and an incremental algorithm for generating finite lattices of given size. In addition to that, we employ a new heuristic non-isomorphism test together with an exact isomorphism test to determine whether a given lattice has already been computed. We start the section by introducing a representation of finite lattices.

3.1 Representation of finite lattices

This section describes the representation of lattices used in our algorithms. Consider a finite lattice $\mathbf{L} = \langle L, \leq \rangle$ where $|L| = n$. We represent $\mathbf{L}$ by an adjacency matrix of $\leq$, i.e. by an $n \times n$ matrix with rows and columns labeled by elements of L and entries containing 1 or 0. For a row and a column labeled by a and b, the corresponding entry is 1 if $a \leq b$ and 0 if $a \not\leq b$. An adjacency matrix for $\mathbf{L} = \langle L, \leq \rangle$ is not unique, due to possible permutations of row and columns. We use upper triangular adjacency matrices. Due to the following theorem, every adjacency matrix can be transformed into an upper triangular form by choosing an appropriate linear order $\preccurlyeq$:

Theorem 1 (see, e.g., [3, 19])**.** *Let $\mathbf{L} = \langle L, \leq \rangle$ be a finite lattice. Then there exists a linear order $\preccurlyeq$ on L which extends $\leq$, i.e. for every $a, b \in L$, if $a \leq b$ then $a \preccurlyeq b$.* □

Thus, we can consider an upper triangular adjacency matrix in which rows and columns are listed in the order given by $\preccurlyeq$. Fig. 1 shown a Hasse diagram of a finite lattice with $L = \{0, a, b, c, d, 1\}$. An adjacency matrix whose rows and columns are ordered by a linear order $\preccurlyeq$ extending $\leq$ such that $0 \preccurlyeq a \preccurlyeq b \preccurlyeq c \preccurlyeq d \preccurlyeq 1$ is depicted in the left part of Fig. 1 (for better readability we replaced 1s by "crosses" and 0s by "blanks"). It is easy to see that if $\preccurlyeq$ is an extension of $\leq$, the corresponding adjacency matrix is upper triangular.

For $|L| = n$, the upper triangle contains $\frac{n(n+1)}{2}$ entries but not all of them carry an essential information. Namely, since $0 \leq a$, $a \leq a$, and $a \leq 1$, for each $a \in L$, we can ignore the corresponding entries (diagonal, first row, and last column). The remaining inner triangle, which in our case is the gray area in Fig. 1, still uniquely represents $\leq$. For an n-element lattice, the inner triangle contains

$$\max\left(0,\, 3 + \frac{n(n-5)}{2}\right) \tag{4}$$

entries. Such entries can be encoded by a binary vector of length (4). For instance, the lattice from Fig. 1 can be represented by a binary vector 001110 (i.e., by a concatenation of binary vectors 001, 11, and 0 encoding the bits from rows a, b, and c of the grey area of the adjacency matrix). The following theorem asserts universality of representing n-element lattices by lattice orders $\leq$ on a fixed set L with a fixed linear order $\preccurlyeq$ which extends $\leq$.

Theorem 2. *Fix an n-element set L and a linear order $\preccurlyeq$ on L. Then for every n-element lattice $\mathbf{L}' = \langle L', \leq' \rangle$ there is a lattice order $\leq$ on L such that*

(i) *$\preccurlyeq$ extends $\leq$ and*
(ii) *$\mathbf{L}' = \langle L', \leq' \rangle$ is isomorphic to $\mathbf{L} = \langle L, \leq \rangle$.*

Proof. Denote $L = \{a_1, \ldots, a_n\}$ and $L' = \{a'_1, \ldots, a'_n\}$ and assume $a_1 \preccurlyeq a_2 \preccurlyeq \cdots \preccurlyeq a_n$. Take a linear order $\preccurlyeq'$ that extends $\leq'$. We can write $a'_{i_1} \preccurlyeq' a'_{i_2} \preccurlyeq' \cdots \preccurlyeq' a'_{i_n}$, where $\{i_1, \ldots, i_n\} = \{1, \ldots, n\}$. Define a map $h\colon L \to L'$ by $h(a_j) = a'_{i_j}$ $(j = 1, \ldots, n)$ and a binary relation $\leq$ on L by $a_j \leq a_k$ iff $a'_{i_j} \leq' a'_{i_k}$. Clearly, h is an isomorphism between $\langle L', \leq' \rangle$ and $\langle L, \leq \rangle$. □

Due to Theorem 2, every n-element lattice can be represented by a binary vector encoding the inner triangle of an adjacency matrix of a lattice order on a fixed set L with a fixed linear order $\preccurlyeq$. A natural choice is $L = \{1, \ldots, n\}$ and $1 \preccurlyeq 2 \preccurlyeq \cdots \preccurlyeq n$. Clearly, not every binary vector encoding the inner triangle of an adjacency matrix represents a lattice order.

Procedure *meet*(*leq, i, j*)

```
   Data: two-dimensional array leq (dimensions n × n); 0 ≤ i < j ≤ n − 1
   Result: infimum of elements given by indices i and j
1  if leq[i, j] = 1 then
2  |  return i
3  else
4  |  for k from i − 1 downto 0 do
5  |  |  if leq[k, i] = 1 and leq[k, j] = 1 then
6  |  |  |  return k
7  |  |  end
8  |  end
9  end
```

Procedure *join*(*leq, i, j*)

```
   Data: two-dimensional array leq (dimensions n × n); 0 ≤ i < j ≤ n − 1
   Result: supremum of elements given by indices i and j
1  if leq[i, j] = 1 then
2  |  return j
3  else
4  |  for k from j + 1 upto n − 1 do
5  |  |  if leq[i, k] = 1 and leq[j, k] = 1 then
6  |  |  |  return k
7  |  |  end
8  |  end
9  end
```

Computational Issues A representation of a lattice by a binary vector encoding the inner triangle of an adjacency matrix has several advantages. It is concise and makes possible an efficient computation of $\leq$, $\wedge$, $\vee$, and transitive closures. For instance, $a \leq b$ can be checked in a constant time by accessing the corresponding adjacency matrix entry. Moreover, the upper triangular form ensures that if $a \prec b$ (i.e., if $a \preccurlyeq b$ and $a \neq b$), then $b \not\leq a$. Hence, $b \not\leq a$ can sometimes be decided even without accessing adjacency matrix. Suprema and infima can be computed with asymptotic time complexity $O(n)$, where $n = |L|$. Consider $L = \{a_0, \ldots, a_{n-1}\}$ and $\preccurlyeq$ such that $a_0 \preccurlyeq a_1 \preccurlyeq \cdots \preccurlyeq a_{n-1}$. If $\leq$ (represented by an upper triangular adjacency matrix) is a lattice order, infima and suprema can be computed by procedures *meet* and *join*. Both the procedures accept two-dimensional array *leq* representing the adjacency matrix and indices i and j ($i < j$) of elements in L as input arguments. The returned values are indices of the infimum and supremum of the elements, respectively. Both the algorithms are sound:

Proof of soundness of meet(*leq, i, j*) *and join*(*leq, i, j*). We examine *meet* because the soundness of *join* can be justified analogously. Lines 1–3 check whether $a_i \leq a_j$ in which case i is returned. Otherwise (i.e., if $a_i \not\leq a_j$), the infimum is found among elements $a_0, \ldots, a_{i-1}$ in a loop (lines 4–9). The loop goes from $k = i - 1$ down to 0. The greatest index k for which $a_k \leq a_i$ and $a_k \leq a_j$ is returned (line 6). Such a_k is the greatest lower bound. Indeed, $a_k \in \mathcal{L}(\{a_i, a_j\})$ due to line 5. Since a_k is an element with the greatest k satisfying $a_k \in \mathcal{L}(\{a_i, a_j\})$, no a_l with $l < k$ can be strictly greater than a_k because $\preccurlyeq$ extends $\leq$. This shows that a_k is the greatest lower bound of a_i and a_j. □

Note that due to our representation of $\leq$, it is not necessary to compute the cones given by (1) and (2) and then determine their greatest and least elements—both the tasks are done simultaneously in less than n elementary steps.

Procedure *transitive-closure*(*leq*)

```
    Data: two-dimensional array leq (dimensions n × n)
    Result: transitive closure of relation represented by leq
 1  for i from 1 upto n − 4 do
 2      for j from i + 1 upto n − 3 do
 3          if leq[i, j] = 1 then
 4              for k from j + 1 upto n − 2 do
 5                  if leq[j, k] = 1 then
 6                      set leq[i, k] to 1
 7                  end
 8              end
 9          end
10      end
11  end
```

Another problem we need to deal with is computing transitive closures of relations represented by upper triangular adjacency matrices. As we discuss later, when generating lattices, we modify adjacency matrices of transitive relations by updating selected entries from 0 to 1. After this operation, we compute the transitive closure of the relation to ensure that the adjacency matrix represents a partial order. An algorithm for computing a transitive closure is described in procedure *transitive-closure*. The procedure accepts an upper triangular adjacency matrix and alters it by inserting 1s. The procedure is sound:

Proof of soundness of procedure transitive-closure (*leq*). Our procedure is a simplified version of the usual algorithm for computing transitive closures. Let leq, leq^*, and leq^c denote the original input relation, its transitive closure, and the relation produced by procedure *transitive-closure* (we tacitly identify the relations with their adjacency matrices). Clearly, if leq is transitive, line 6 does not change any nonzero entry to 1, i.e. leq is not modified and we have $leq = leq^* = leq^c$. Otherwise, there are $1 \leq i < j < k \leq n-2$ such that $leq\,[i,j] = 1$, $leq\,[j,k] = 1$, and $leq\,[i,k] = 0$. Therefore, line 6 changes at least one zero entry in leq to 1. Obviously, $leq \subset leq^c \subseteq leq^*$. Hence, is suffices to prove that $leq^* \subseteq leq^c$. Consider indices p and q such that $leq^*[p,q] = 1$ and $leq\,[p,q] = 0$ (a nontrivial case). It remains to show that $leq^c[p,q] = 1$. Let $i \sqsubset k$ denote the fact that $leq\,[i,k] = 1$ and there is no j such that $i < j < k$, $leq\,[i,j] = 1$, and $leq\,[j,k] = 1$. Since $leq^*[p,q] = 1$ there are indices $p = j_1 \sqsubset j_2 \sqsubset \cdots \sqsubset j_l = q$. Now, consider the three nested loops between lines 1–11. Let $i = p = j_1$, $j = j_2$, and $k = j_3$. Since $leq\,[i,j] = leq\,[j_1,j_2] = 1$ and $leq\,[j_2,j_3] = 1$, both if-conditions (lines 3 and 5) are true and line 6 will set $leq\,[i,k] = leq\,[j_1,j_3]$ to 1. When the computation reaches configuration $i = p = j_1$, $j = j_3$, and $k = j_4$, we already have $leq\,[i,j] = leq\,[j_1,j_3] = 1$ from the previous step and $leq\,[j,k] = leq\,[j_3,j_4] = 1$, i.e. line 6 will set $leq\,[i,k] = leq\,[j_1,j_4]$ to 1. By induction, $leq\,[p,q] = leq\,[j_1,j_l]$ will be set to 1 after finitely many steps. As a consequence, $leq^c[p,q] = 1$, finishing the proof. □

3.2 Characteristic vectors of finite lattices

In this section, we consider properties of lattices which we use in heuristic tests, described in Section 3.3, to quickly recognize non-isomorphic lattices. The properties are shared by isomorphic lattices but are highly unlikely to be shared by non-isomorphic lattices. The properties of a given lattice are encoded by a characteristic vector which we now describe.

Consider a finite lattice $\mathbf{L} = \langle L, \leq \rangle$ and a linear order $\preccurlyeq$ extending $\leq$. We define

$$\mathcal{P}(L) = \big\{\{a,b\} \,|\, a,b \in L,\ a \neq 0,\ b \neq 1, \text{ and } a \neq b\big\}. \tag{5}$$

For each $a \in L$, we define four non-negative integers $v_1(a), \ldots, v_4(a)$, characterizing some of the properties of a:

$$v_1(a) = |\mathcal{L}(\{a\})| = |\{b \in L \,|\, b \leq a\}|, \tag{6}$$
$$v_2(a) = |\mathcal{U}(\{a\})| = |\{b \in L \,|\, a \leq b\}|, \tag{7}$$
$$v_3(a) = |\{A \in \mathcal{P}(L) \,|\, a = \textstyle\bigwedge A\}|, \tag{8}$$
$$v_4(a) = |\{A \in \mathcal{P}(L) \,|\, a = \textstyle\bigvee A\}|, \tag{9}$$

and put

$$v(a) = \langle v_1(a), v_2(a), v_3(a), v_4(a) \rangle. \tag{10}$$

By definition, $v_1(a)$ and $v_2(a)$ represent the numbers of elements which are less/greater than or equal to a. Values of $v_3(a)$ and $v_4(a)$ represent the numbers of pairs of elements from L whose infimum/supremum gives a. Roughly speaking, $v_1(a), \ldots, v_4(a)$ represent a "position" of $a \in L$ in the lattice. Note that $v(a)$ depends on the lattice order $\leq$ on L, i.e. two different $\leq_1$ and $\leq_2$ on L can yield different $v(a)$s. In order to make $\mathbf{L} = \langle L, \leq \rangle$ explicit, we denote $v_i(a)$ and $v(a)$ by $v_i^{\mathbf{L}}(a)$ and $v^{\mathbf{L}}(a)$.

Example 1. Recall the finite lattice from Fig. 1. The values of v_is are shown in the following table:

$\mathbf{L}$	0	a	b	c	d	1
v_1	1	2	2	3	4	6
v_2	6	3	4	2	2	1
v_3	2	1	3	0	0	0
v_4	0	0	0	1	3	2

That is, $v_1(0) = 1$, $v_2(0) = 6$, $v_3(0) = 2$, $v_4(0) = 0$, i.e. $v(0) = \langle 1, 6, 2, 0 \rangle$; etc. $v_3(b) = 3$ means there are exactly three subsets in $\mathcal{P}(L)$ whose infimum yields b, namely: $\{b, c\}$, $\{b, d\}$, and $\{c, d\}$.

In order to determine equality of vectors $v(a)$ (for all $a \in L$), we sort them according to their lexical ordering. That is, we define a *lexical* (*linear*) *order* $\leqslant_{\text{lex}}$ on four-tuples of integers as follows. For $x = \langle x_1, x_2, x_3, x_4 \rangle \in \mathbb{Z}^4$ and $y = \langle y_1, y_2, y_3, y_4 \rangle \in \mathbb{Z}^4$ we put $x \leqslant_{\text{lex}} y$ iff either $x = y$ (tuples are identical) or there is $i \in \{1, \ldots, 4\}$ such that, for each $j < i$, $x_j = y_j$ and $x_i < y_i$.

Definition 1 (characteristic vector)**.** A *characteristic vector of a finite lattice* $\mathbf{L} = \langle L, \leq \rangle$ is a vector of integers which results by concatenating vectors $v(a)$ $(a \in L)$ listed in the lexical order $\leqslant_{\text{lex}}$.

Example 2. In case of the lattice from Fig. 1, we can see that

$$v(0) \leqslant_{\text{lex}} v(a) \leqslant_{\text{lex}} v(b) \leqslant_{\text{lex}} v(c) \leqslant_{\text{lex}} v(d) \leqslant_{\text{lex}} v(1).$$

That is, the characteristic vector is a concatenation of vectors $v(0)$, $v(a)$, $v(b)$, $v(c)$, $v(d)$, and $v(1)$:

$$\langle 1, 6, 2, 0, 2, 3, 1, 0, 2, 4, 3, 0, 3, 2, 0, 1, 4, 2, 0, 3, 6, 1, 0, 2 \rangle.$$

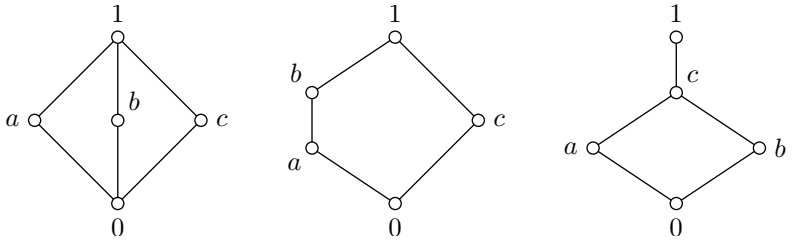
Example 3. Fig. 2 shows all five-element lattices and the corresponding tables describing values of v_is. The columns of the tables are already listed in the lexical order $\leqslant_{\text{lex}}$. For instance, the characteristic vector of $\mathbf{L}_{5,2}$ is given by a concatenation of $v(0)$, $v(c)$, $v(a)$, $v(b)$, and $v(1)$, i.e.

$$\langle 1, 5, 2, 0, 2, 2, 0, 0, 2, 3, 1, 0, 3, 2, 0, 1, 5, 1, 0, 2 \rangle.$$

Determining the characteristic vector of a given n-element lattice can be solved with an asymptotic complexity of $O(n^3)$. Indeed, traversing trough the binary vector representing the adjacency matrix is done in $O(n^2)$ steps, each such a step requires a computation of infima and suprema, which can be done in $O(n)$ steps. Thus, we need $O(n^3)$ steps to find the values of all $v(a)$s. Finally, an efficient sorting algorithm like heap-sort can be used to sort $v(a)$s according to $\leqslant_{\text{lex}}$ in $O(n \log n)$ steps which does not increase the asymptotic complexity. Thus, the overall time complexity of determining the characteristic vector is $O(n^3)$.

3.3 Heuristic tests of non-isomorphism

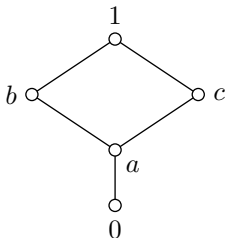
A direct procedure to test whether two n-element lattices are isomorphic, given by the definition of an isomorphism, leads to generating $n!$ bijective maps between two n-element lattices and checking whether some of them is an isomorphism. In this section we propose a procedure which quickly disqualifies most of non-isomorphic lattices. In general, it cannot be used to decide whether two lattices *are* isomorphic. In the latter case, we use a brute force checking of bijections between lattices. However, the advantage of our procedure is that even if we are compelled to check the bijections, we can restrict ourselves only to "isomorphism candidates". The number of isomorphism candidates is in most cases much smaller than $n!$. We start with the following obvious theorem.



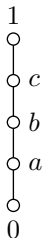
$\mathbf{L}_{5,1}$	0	a	b	c	1
v_1	1	2	2	2	5
v_2	5	2	2	2	1
v_3	3	0	0	0	0
v_4	0	0	0	0	3

$\mathbf{L}_{5,2}$	0	c	a	b	1
v_1	1	2	2	3	5
v_2	5	2	3	2	1
v_3	2	0	1	0	0
v_4	0	0	0	1	2

$\mathbf{L}_{5,3}$	0	a	b	c	1
v_1	1	2	2	4	5
v_2	5	3	3	2	1
v_3	1	1	1	0	0
v_4	0	0	0	3	0



$\mathbf{L}_{5,4}$	0	a	b	c	1
v_1	1	2	3	3	5
v_2	5	4	2	2	1
v_3	0	3	0	0	0
v_4	0	0	1	1	1



$\mathbf{L}_{5,5}$	0	a	b	c	1
v_1	1	2	3	4	5
v_2	5	4	3	2	1
v_3	0	2	1	0	0
v_4	0	0	1	2	0

Figure 2: All five-element lattices (up to isomorphism) and their characteristic vectors written in tables.

Theorem 3. *Let* $\mathbf{L}_1$ *and* $\mathbf{L}_2$ *be lattices,* $h\colon L_1 \to L_2$ *be a lattice isomorphism. Then, for each* $a \in L_1$*,*

$$v^{\mathbf{L}_1}(a) = v^{\mathbf{L}_2}\big(h(a)\big). \tag{11}$$

As a consequence, two isomorphic finite lattices have the same characteristic vectors.

Hence, any two finite lattices with different characteristic vectors are not isomorphic. Thus, a *quick non-isomorphism test* can be performed by checking the inequality of characteristic vectors. If the characteristic vectors of $\mathbf{L}_1$ and $\mathbf{L}_2$ are equal, we cannot tell whether $\mathbf{L}_1 \cong \mathbf{L}_2$ and need further analysis. Suppose the characteristic vectors of $\mathbf{L}_1$ and $\mathbf{L}_2$ are equal and both $\mathbf{L}_1$ and $\mathbf{L}_2$ are defined on the same universe set L linearly ordered by a fixed $\preccurlyeq$. In order to confirm/deny that $\mathbf{L}_1$ and $\mathbf{L}_2$ are isomorphic, it is not necessary to go through all permutations of L (bijections $h\colon L \to L$) because Theorem 3 says that the elements corresponding under any isomorphism of $\mathbf{L}_1$ and $\mathbf{L}_2$ must have the same values of $v(\cdots)$, see (11). Thus, it suffices to generate and check only permutations satisfying (11). We call such permutations isomorphism candidates:

Definition 2 (isomorphism candidates)**.** Let $\mathbf{L}_1 = \langle L, \leq_1 \rangle$ and $\mathbf{L}_2 = \langle L, \leq_2 \rangle$. A permutation $h\colon L \to L$ is called an *isomorphism candidate* if (11) is satisfied for each $a \in L_1$.

Example 4. Consider the lattice from Fig. 1 and its characteristic vector from Example 1. Suppose we arrive at a lattice with the same characteristic vector. Since all columns of the table in Example 1 are pairwise distinct, to decide whether the two lattices are isomorphic, it suffices to check just one permutation (just one isomorphism candidate).

Example 5. In case of lattice $\mathbf{L}_{5,3}$ from Fig. 2 we would have to check two isomorphism candidates, because two columns in the corresponding table are equal. In case of $\mathbf{L}_{5,1}$, we would have to check $3! = 6$ candidates, because three columns of the table are equal, etc. One can see that in case of five-element lattices, the non-isomorphic lattices have pairwise different characteristic vectors. Hence, no checking of candidates is necessary. However, for lattices with $|L| \geq 8$, there are situations when the heuristic test fails as we show later. Therefore, checking all isomorphism candidates is necessary if $|L| \geq 8$.

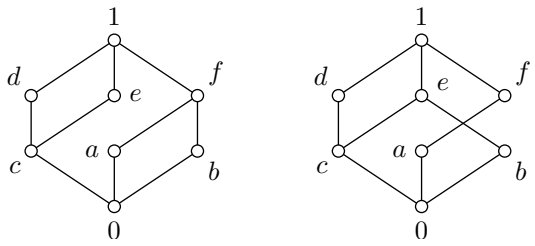


Figure 3: Eight element lattices that fail the heuristic non-isomorphism test.

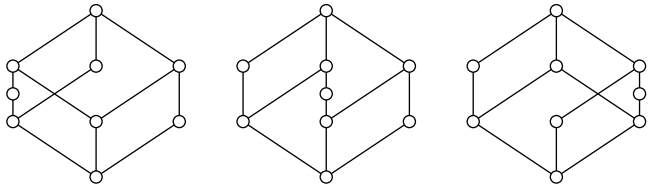


Figure 4: Nine element lattices that fail the heuristic non-isomorphism test.

The heuristic test of non-isomorphism takes two lattices $\mathbf{L}_1 = \langle L, \leq_1 \rangle$ and $\mathbf{L}_2 = \langle L, \leq_2 \rangle$ as its input, and outputs "true" (may or may not be isomorphic) or "false" (not isomorphic):

Algorithm 1 (heuristic test of non-isomorphism)**.**

(i) Count 1's in the binary vector encoding $\mathbf{L}_1$ and in the binary vector encoding $\mathbf{L}_2$. If the numbers of 1's in both the vectors are different, return "false". Otherwise go to step (ii).

(ii) Determine characteristic vectors of $\mathbf{L}_1$ and $\mathbf{L}_2$. Return "true" if the vectors coincide; return "false" otherwise.

Denote the output of the heuristic test for $\mathbf{L}_1$ and $\mathbf{L}_2$ by $\mathrm{Iso}_H(\mathbf{L}_1, \mathbf{L}_2)$. If $\mathrm{Iso}_H(\mathbf{L}_1, \mathbf{L}_2)$ is "false" then, according to our previous observations, $\mathbf{L}_1$ and $\mathbf{L}_2$ are not isomorphic. If $\mathrm{Iso}_H(\mathbf{L}_1, \mathbf{L}_2)$ is "true", we employ an exact test of isomorphism. The exact test accepts as its input the lattices and their characteristic vectors (computed by the previous use of the heuristic test). The output of the exact test is "false" (not isomorphic) or "true" (isomorphic). The exact test proceeds as follows:

Algorithm 2 (exact test of isomorphism)**.**

(i) Initialize the generator of isomorphism candidates given by the characteristic vector. Proceed with step (ii).

(ii) If there are no further isomorphism candidates left for checking, return "false". Otherwise, take the next isomorphism candidate h and go to step (iii).

(iii) For each $a, b \in L$, check condition (3). If the condition is true for each $a, b \in L$, return "true". Otherwise, skip h (i.e., remove h from the list of isomorphism candidates) and go to step (ii).

Now, we consider the situations when a *failure of the heuristic test* occurs, i.e. when $\mathrm{Iso}_H(\mathbf{L}_1, \mathbf{L}_2)$ is "true" for *non-isomorphic* lattices $\mathbf{L}_1$ and $\mathbf{L}_2$, in which case the heuristic test alone is not sufficient to decide whether $\mathbf{L}_1$ and $\mathbf{L}_2$ are isomorphic. We explore the frequency of such failures. First, let us show that such situations do occur.

Example 6. Consider the eight-element lattices from Fig. 3 and denote them by $\mathbf{L}_1$ and $\mathbf{L}_2$ (from left to right). Both the lattices have the same characteristic vector. Namely, the following table contains entries of the characteristic vector:

$\mathbf{L}_1$	0	a	b	c	d	e	f	1
$\mathbf{L}_2$	0	a	b	c	d	f	e	1
v_1	1	2	2	2	3	3	4	8
v_2	8	3	3	4	2	2	2	1
v_3	10	1	1	3	0	0	0	0
v_4	0	0	0	0	1	1	4	10

	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	2	5	15	53	220	1049	5682	34502	232070
2	0	0	0	0	0	0	0	1	13	125	1159	10963
3	0	0	0	0	0	0	0	0	1	18	212	2035
4	0	0	0	0	0	0	0	0	0	2	28	388
5	0	0	0	0	0	0	0	0	0	0	6	102
6	0	0	0	0	0	0	0	0	0	0	4	65
7	0	0	0	0	0	0	0	0	0	0	0	16
8	0	0	0	0	0	0	0	0	0	0	0	6
10	0	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	2
16	0	0	0	0	0	0	0	0	0	0	0	1

Table 1: Numbers of characteristic vectors of given orders.

As we can see, two pairs of elements have the same columns, i.e. the exact test of isomorphism would go through $2! \cdot 2! = 4$ isomorphism candidates. Each of the four candidates is a map $h\colon L \to L$ where $h(0) = 0$, $\{h(a), h(b)\} = \{a, b\}$, $h(c) = c$, $\{h(d), h(e)\} = \{d, f\}$, $h(f) = e$, and $h(1) = 1$. Thus, we either have $h(a) = a$ and $h(b) = b$, or $h(a) = b$ and $h(b) = a$. In either case, h cannot be an isomorphism: (i) if $h(a) = a$, then $a \leq_1 f$ and $h(a) = a \not\leq_2 e = h(f)$, which violates (3); (ii) if $h(a) = b$, then $h(b) = a$, i.e. $b \leq_1 f$ and $h(b) = a \not\leq_2 e = h(f)$, which again violates (3). Hence, $\mathbf{L}_1$ and $\mathbf{L}_2$ are not isomorphic while having the same characteristic vector. Note that $\mathbf{L}_1$ and $\mathbf{L}_2$ are the only eight-element lattices (up to isomorphism) that fail the heuristic test and, at the same time, are the smallest lattices that fail the test. Fig. 4 shows three pairwise non-isomorphic nine-element lattices which share the same characteristic vector. Hence, any two distinct lattices of the three depicted in Fig. 4 would fail the heuristic test.

Remark 1. Is a failure of the heuristic test rare? We have investigated this problem for lattices with up to 12 elements. Suppose c is a characteristic vector c of a finite lattice. By an *order of c,* denoted $||c||$, we mean the number of pairwise non-isomorphic lattices whose characteristic vector is exactly c. If $||c|| = 1$, there is just one finite lattice (up to isomorphism) with c in which case the heuristic test does not fail. For small lattices, $||c|| = 1$ for every characteristic vector, i.e. the isomorphism can be decided by the heuristic test. Table 1 shows the numbers of characteristic vectors of given orders. The columns of the table correspond to sizes of lattices, the rows correspond to orders of characteristic vectors, and the table entries show how many characteristic vectors (of orders given by rows and sizes given by columns) there are. As mentioned above, for $n \leq 7$, there are only vectors of order 1.

To sum up, for $|L| \leq 7$ the heuristic test never fails. For $|L| \geq 8$ the heuristic test can fail but we can use the information present in characteristic vectors to avoid brute-force checking of all possible bijections between two lattices. Later, we show the numbers of candidates being checked during the isomorphism tests and will see that on average our approach significantly reduces the numbers of bijections needed to decide the isomorphism of lattices.

3.4 Generation of finite lattices: brute-force algorithm

This section describes a brute-force algorithm for generating lattices which employs the heuristic test described in the previous section. A more efficient algorithm derived from this algorithm is presented in the next section.

Non-isomorphic finite lattices of a given size n can be generated the following way. First, we create an *initial lattice* which is encoded by a binary vector (see Section 3.1) containing all 0s and add it to a list of generated lattices. The partially ordered set given by this binary vector is a lattice which, for $|L| = n$, contains an antichain of $n - 2$ elements (i.e., $n - 2$ elements of the lattice are incomparable). Then, we go through all possible binary vectors of the given length. For every vector we check if it represents a lattice which has not yet been generated. If so, we add the lattice to the set of generated lattices and continue with the next binary vector. The procedure goes on until we generate the *final lattice,* which is the lattice encoded by the binary vector full of 1s. Such a vector encodes an n-element chain. During the procedure, we use the isomorphism tests described in Section 3.3. Our method of generation of lattices is described by the following recursive procedure:

Algorithm 3 (brute-force generating of finite lattices)**.**

(i) Set binary vector so that it represents the initial n-element lattice.

(ii) Check if the current binary vector represents a lattice order. If yes, go to step (iii). Otherwise, go to step (iv).

(iii) Check if $\leq$ (lattice order represented by the current binary vector) is isomorphic to a lattice which has already been generated—use the heuristic and exact tests described in Section 3.3.

- If $\leq$ is not isomorphic to any of the generated lattices, add $\leq$ to the set of generated lattices and go to step (iv).
- If $\leq$ is isomorphic to some previously generated $\leq'$ and if in addition $\leq$ equals $\leq'$ (i.e., $\leq$ has already been found), then end this branch of recursion.
- Otherwise (i.e., $\leq$ is isomorphic to some previously generated $\leq'$ but $\leq$ differs from $\leq'$), go to step (iv).

(iv) Loop over all values of the binary vector $\leq$ which are equal to 0; for each of them, perform the following steps one by one:

- Make a copy $\leq'$ of $\leq$ (copy of binary vectors).
- Set to 1 the current value in $\leq'$ which equals 0.
- Compute the transitive closure of $\leq'$ (using procedure *transitive-closure* from Section 3.1).
- Recursively call (ii) for $\leq'$.

After the loop finishes, halt computation.

Proof of soundness of Algorithm 3. Soundness of the algorithm directly follows from the fact that the algorithm goes over all possible upper triangular adjacency matrices represented by binary vectors. For all isomorphic n-element lattices, the algorithm stores exactly one of them, due to step (iii). □

Remark 2. Our test of isomorphism which is based on the heuristic and exact tests seems to be very efficient. For example, generating 9-element lattices using our isomorphism tests took under 2 minutes while the same algorithm which uses only the exact test needed over 8 hours.

3.5 Generation of finite lattices: incremental algorithm

We were able to use the brute-force algorithm to generate lattices with at most 10 elements. We now describe a more efficient algorithm derived from the brute-force one. The algorithm uses lattices with n elements to generate all lattices with $n+1$ elements. We use the following assertion:

Theorem 4. *Let $\mathbf{L} = \langle L, \leq \rangle$ be a finite lattice with $|L| > 1$, $c \in L$ be a coatom in $\mathbf{L}$. Then $L' = L - \{c\}$ equipped with $\leq'$ which is a restriction of $\leq$ on L' is a lattice which is a $\wedge$-sublattice of $\mathbf{L}$.*

Proof. Clearly, $\leq'$ is a partial order. Because c is a coatom, for any $a, b \in L'$ such that $a \neq 1$, we have

$$\mathcal{L}_{\leq'}(\{a,b\}) = \{x \in L' \mid x \leq' a \text{ and } x \leq' b\} = \{x \in L \mid x \leq a \text{ and } x \leq b\} = \mathcal{L}_{\leq}(\{a,b\}).$$

As a consequence, $\mathcal{L}_{\leq'}(\{a,b\})$ has a greatest element which is the infimum of $\{a,b\}$ in $\mathbf{L}'$. If both $a = b = 1$ then obviously $\mathcal{L}_{\leq'}(\{a,b\}) = L'$ and $\mathcal{L}_{\leq}(\{a,b\}) = L$. Thus, the infimum of $a, b \in L'$ in $\mathbf{L}'$ agrees with the infimum of $a, b \in L'$ in $\mathbf{L}$, showing that $\mathbf{L}$ is a $\wedge$-sublattice of $\mathbf{L}$. It remains to show that $\mathbf{L}$ is a $\vee$-semilattice. Take any $a, b \in L'$. We distinguish several situations according to

$$\mathcal{U}_{\leq}(\{a,b\}) = \{x \in L \mid a \leq x \text{ and } b \leq x\}.$$

If $c \notin \mathcal{U}_{\leq}(\{a,b\})$ then $\mathcal{U}_{\leq}(\{a,b\}) = \mathcal{U}_{\leq'}(\{a,b\})$, i.e. the supremum of $\{a,b\}$ in $\mathbf{L}'$ exists and equals the supremum of $\{a,b\}$ in $\mathbf{L}$. Suppose $c \in \mathcal{U}_{\leq}(\{a,b\})$. If $a \vee b = c$ (the supremum of $\{a,b\}$ in $\mathbf{L}$ equals c), we must have $\mathcal{U}_{\leq}(\{a,b\}) = \{c,1\}$. Hence, $\mathcal{U}_{\leq'}(\{a,b\}) = \{1\}$, i.e. the supremum of $\{a,b\}$ in $\mathbf{L}'$ equals 1. If $a \vee b \neq c$, the least element of $\mathcal{U}_{\leq}(\{a,b\})$ equals the least element of $\mathcal{U}_{\leq'}(\{a,b\})$, i.e. in this case the supremum of $\{a,b\}$ in $\mathbf{L}'$ equals $a \vee b$ (the supremum of $\{a,b\}$ in $\mathbf{L}$). To sum up, we have shown that $\mathbf{L}' = \langle L', \leq' \rangle$ is a lattice. □

Remark 3. In general $\mathbf{L}' = \langle L', \leq' \rangle$ from Theorem 4 is not a $\vee$-sublattice of $\mathbf{L} = \langle L, \leq \rangle$. For instance, the eight-element Boolean lattice $\mathbf{L} = \langle 2^{\{a,b,c\}}, \subseteq \rangle$ does not contain any seven-element sublattice [19].

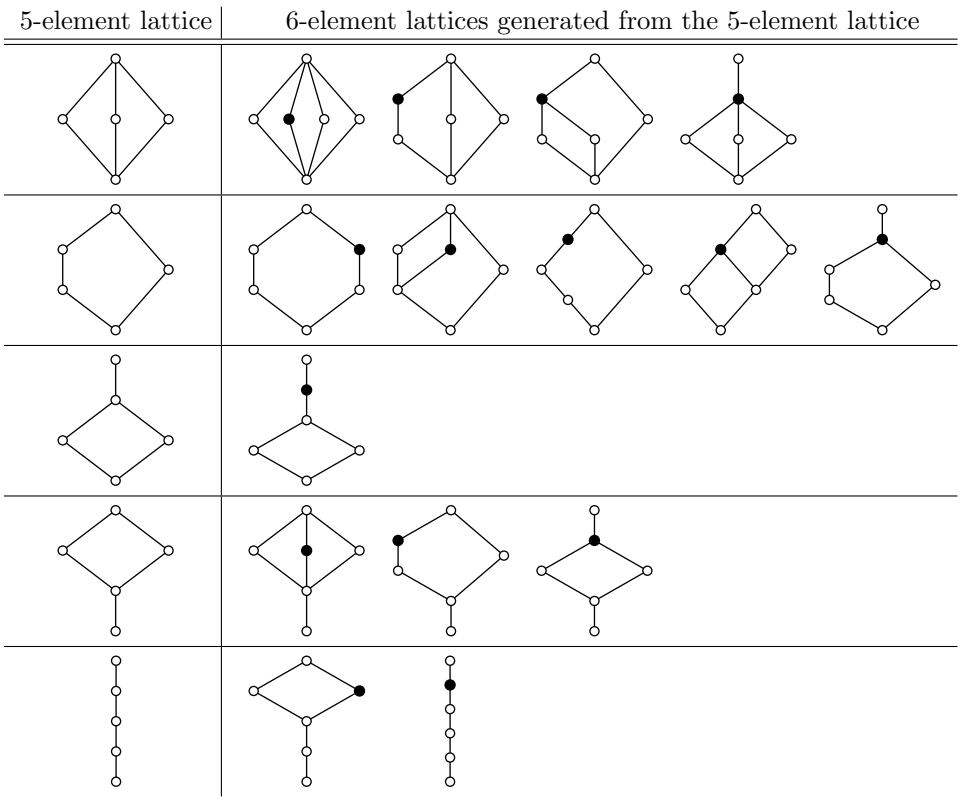


Figure 5: Hasse diagrams of all six-element lattices constructed from five-element lattices.

Theorem 4 can be used as follows. In our representation of finite lattices by upper triangular adjacency matrices, the last nontrivial column (i.e., the column before the one corresponding to 1) represents a coatom. Considering the example from Fig. 1, this is the column corresponding to d. If we "remove" the column and row corresponding to d, we obtain another upper triangular adjacency matrix. According to Theorem 4, this is an adjacency matrix of a 5-element lattice. This lattice is isomorphic to $\mathbf{L}_{5,2}$ from Fig. 2. In general, any adjacency matrix of an n-element lattice contains an adjacency matrix of some $(n-1)$-element lattice. This observation allows us to use n-element lattices to generate $(n+1)$-element lattices simply by adding a column and a row representing a new coatom. This significantly reduces the number of new entries in upper triangular adjacency matrices during the computation.

Example 7. Fig. 5 illustrates how 6-element lattices are generated from 5-element lattices by adding new coatoms. The first column of the table represents a five-element lattice. The remaining lattices in each row represent 6-element lattices which can be constructed from the 5-element lattice the way we have just described. The black nodes in the Hasse diagrams represent new coatoms. Observe that if we remove such coatoms then, according to Theorem 4, we obtain a lattice isomorphic to the 5-element lattice in the left-most column.

The following algorithm accepts as an input a collection of all n-element lattices and produces a list of all $(n+1)$-element lattices. Each of the n-element lattices is used to form an upper triangular adjacency matrix which has the same entries as the adjacency matrix of the n-element lattice plus a new column representing a new coatom. Then, only the values in this new column are considered for generating $(n+1)$-element lattices. The algorithm proceeds as follows:

Algorithm 4 (generating finite lattices incrementally)**.**

(i) If there are no more n-element lattices, halt computation. Otherwise, take first n-element lattice $\mathbf{L}$, remove it from the collection of n-element lattices, and go to (ii).

(ii) Construct a binary vector encoding an upper triangular adjacency matrix for a lattice with $n+1$ elements. Fill values of the vector so that (a) values corresponding to the last nontrivial column of the adjacency matrix (new coatom) are filled with zeros and (b) other values encode the upper triangular adjacency matrix of $\mathbf{L}$. Continue with (iii).

size of L	1	2	3	4	5	6	7	8	9	10	11	12
vector length	0	0	0	1	3	6	10	15	21	28	36	45
possible relations	2^0	2^0	2^0	2^1	2^3	2^6	2^{10}	2^{15}	2^{21}	2^{28}	2^{36}	2^{45}
generated lattices	1	1	1	2	8	37	201	1273	9677	86105	887112	10406211
tests of isomorphism	0	0	0	0	3	22	148	1055	8661	80921	859881	10277785
generated candidates	0	0	0	0	3	22	155	1158	10054	97113	1058787	12765905
ratio	–	–	–	–	1.00	1.00	1.05	1.10	1.16	1.20	1.23	1.24
non-isomorphic lattices	1	1	1	2	5	15	53	222	1078	5994	37622	262776

Table 2: Lengths of vectors encoding finite lattices up to 12 elements.

(iii) Check if the current binary vector represents a lattice order. If yes, go to step (iv). Otherwise, go to step (v).

(iv) Check if $\leq$ (lattice order represented by the current binary vector) is isomorphic to a lattice which has already been generated—use the heuristic and exact tests described in Section 3.3.

- If $\leq$ is not isomorphic to any of the generated lattices, add $\leq$ to the set of generated lattices and go to step (v).
- If $\leq$ is isomorphic to some previously generated $\leq'$ and if in addition $\leq$ equals $\leq'$ (i.e., $\leq$ has already been found), then end this branch of recursion.
- Otherwise (i.e., $\leq$ is isomorphic to some previously generated $\leq'$ but $\leq$ differs from $\leq'$), go to step (v).

(v) Loop over all values of the binary vector $\leq$ which are equal to 0 and which correspond to the last nontrivial column of the upper triangular adjacency matrix. For each of them, perform the following steps one by one:

- Make a copy $\leq'$ of $\leq$ (copy of binary vectors).
- Set to 1 the current value in $\leq'$ which equals 0.
- Compute the transitive closure of $\leq'$ (see procedure *transitive-closure* from Section 3.1).
- Recursively call (iii) for $\leq'$.

After the loop finishes, go to step (i).

Proof of soundness of Algorithm 4. Soundness follows from the soundness of Algorithm 3 and from Theorem 4. □

Remark 4. Since the loop contained in (v) of Algorithm 4 goes just over the entries corresponding to the last nontrivial column of the adjacency matrix, we have significantly reduced the search space in which we look for finite lattices.

Remark 5. An interesting thing to note is the average number of isomorphism candidates that are used during each isomorphism test. Recall that when the heuristic test is positive, we must decide the isomorphism by finding an appropriate bijection between lattices (this is a part of the exact test). Using characteristic vectors, we can restrict ourselves to certain bijections only, namely to isomorphism candidates described in Section 3.3. The average number of such isomorphism candidates is surprisingly low: during the generation of 12-element lattices, the algorithm checks approximately 1.24 isomorphism candidates per each isomorphism test. The exact values are depicted in Table 2. Columns of the table correspond to sizes of lattices. The first two rows contain the information about the length of binary vectors encoding adjacency matrices and the total number of all binary vectors of such lengths. The third row contains the numbers of all generated lattices, including the isomorphic ones. The next three lines depict the numbers of exact isomorphism tests performed, the number of isomorphism candidates used in these tests, and their ratio. As we can see, the ratio is close to 1, i.e., on average we need just one isomorphism candidate per isomorphism test. Hence, roughly speaking, if two lattices are isomorphic, the map proving the isomorphism is usually the first candidate considered. Thus, the exact isomorphism test using isomorphism candidates is very efficient for lattices with $|L| \leq 12$.

$\otimes$	0	a_1	a_2	$\cdots$	a_{n-2}	1
0	0	0	0	$\cdots$	0	0
a_1	0					a_1
a_2	0					a_2
$\vdots$	$\vdots$					$\vdots$
a_{n-2}	0					a_{n-2}
1	0	a_1	a_2	$\cdots$	a_{n-2}	1

Figure 6: Initial assignment of lattice values to a table for $\otimes$ and procedure *fill*.

4 Generation of finite residuated lattices

In this section we describe a way to generate residuated lattices of a given size. We describe an algorithm which, for a given finite lattice $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$, generates all pairs $\langle \otimes, \rightarrow\rangle$ of adjoint operations on $\mathbf{L}$. The algorithm from Section 3 and this algorithm provide us with an algorithm for generating residuated lattices up to a given size.

4.1 Representation of finite residuated lattices

Let $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$ be a finite lattice. By definition of a residuated lattice, we are looking for all couples $\langle \otimes, \rightarrow\rangle$ of operations such that $\langle L, \otimes, 1\rangle$ is a commutative monoid, and $\otimes$ and $\rightarrow$ satisfy adjointness. Note that not all finite lattices admit adjoint operations $\otimes$ and $\rightarrow$. The algorithm which is described later in this section generates only $\otimes$ (multiplication, truth function of "fuzzy conjunction") and tests a condition which is equivalent to the existence of $\rightarrow$ (residuum, truth function of "fuzzy implication") satisfying adjointness with $\otimes$. Namely, we take advantage of the following assertion:

Theorem 5. *Let $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$ be a finite lattice, $\langle L, \otimes, 1\rangle$ be a commutative monoid such that $\otimes$ is monotone w.r.t. $\leq$. Then the following are equivalent:*

(i) *there exists (unique) $\rightarrow$ satisfying adjointness w.r.t. $\otimes$;*

(ii) *for each $a, b, c \in L$: $a \otimes (b \vee c) = (a \otimes b) \vee (a \otimes c)$;*

(iii) *$\rightarrow$ given by $a \rightarrow b = \bigvee\{c \in L \,|\, a \otimes c \leq b\}$ satisfies adjointness w.r.t. $\otimes$.*

Proof. Follows from finiteness of L and properties of residuated lattices. The equivalence if (i) and (iii) is shown, e.g., in [4]. In general, (i) is true iff $a \otimes \bigvee_{i\in I} b_i = \bigvee_{i\in I}(a \otimes b_i)$ holds for any $a, b_i \in L$ ($i \in I$), see [4]. Since L is finite, the latter condition is equivalent to (ii). □

Due to Theorem 5, it suffices to generate all monotone, commutative, and associative operations $\otimes$ for which 1 (greatest element of $\mathbf{L}$) is a neutral element and which satisfy condition (ii) of Theorem 5. If $\otimes$ satisfies all these conditions, we can use (iii) to compute the residuum $\rightarrow$ of $\otimes$, which is uniquely given.

The basic idea of our algorithm is that we systematically go through all candidates $\otimes$ which may satisfy assumptions of Theorem 5 and (ii). Let $|L| = n$ and $L = \{0 = a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1} = 1\}$. Furthermore, assume that our indexing extends the lattice order, i.e. that $a_i \leq a_j$ implies $i \leq j$. For $|L| = n$, there are n^{n^2} distinct binary operations on L, which is too large a number even for small n. Generating all of them is not feasible.

The task to find a multiplication can be seen as the task to fill a table as the one in Fig. 6 by lattice values. A table entry given by row i and column j represents the value $a_i \otimes a_j$. Since $\otimes$ needs to be commutative, we can focus only on the upper triangle of the table including the diagonal because in general $\otimes$ is not idempotent. Moreover, some lattice values in the table are the same for all multiplications because the properties of residuated lattices imply

$$a \otimes 0 = 0 \otimes a = 0, \qquad a \otimes 1 = 1 \otimes a = a.$$

The other entries in the table can take any values from $L - \{1\}$. Fortunately, we can restrict the set of possible values for each table entry using the following well-known fact:

Theorem 6. *Let $\mathbf{L}$ be a residuated lattice. Then, for each $a, b \in L$,*

(i) *$a \otimes b \leq a \wedge b$;*

(ii) $\bigvee\{c \otimes d \,|\, c, d \in L \text{ such that } c \leq a \text{ and } d \leq b\} \leq a \otimes b$.

Proof. (i) is a well-known property of residuated lattices, (ii) follows from the monotony of $\otimes$. □

Theorem 6 provides upper and lower bounds for the results of a multiplication. In more detail, for each $a, b \in L$, the upper bound is given by (i). The lower bound can be computed using (ii) before each new assignment. Since we assign lattice values to the table one by one, for a considered pair $a, b \in L$ of lattice values we can take all $c, d \in L$ such that (i) the value $c \otimes d$ is already assigned, and (ii) $c \leq a$ and $d \leq b$. Then we can compute the supremum of all such values $c \otimes d$ which is then the lower bound of $a \otimes b$.

4.2 Algorithm for generating multiplications

In the algorithm, a table for $\otimes$, such as the one in Fig. 6, is filled from its top-left corner to its bottom-right corner. The table entries are traversed in the following order: $a_1 \otimes a_1$, $a_1 \otimes a_2, \ldots, a_1 \otimes a_{n-3}$, $a_1 \otimes a_{n-2}$, $a_2 \otimes a_2$, $a_2 \otimes a_3, \ldots, a_{n-2} \otimes a_{n-2}$. For each new entry being added to the table we check several conditions to see that $\otimes$ represents a "candidate" for multiplication. Namely, for each $a, b, c \in L$ we check

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c, \tag{12}$$

$$a \otimes (b \vee c) = (a \otimes b) \vee (a \otimes c), \tag{13}$$

$$a \leq b \text{ implies } a \otimes c \leq b \otimes c, \tag{14}$$

provided that the expressions in (12)–(14) are defined (recall that we deal with a partial operation $\otimes$ which is being constructed, i.e., some values of $x \otimes y$ may not be defined). If the currently assigned value of $a_i \otimes a_j$ violates the conditions above, then we go back and set $a_i \otimes a_j$ to another value. Otherwise we move to the next blank position in the table and compute possible values of the multiplication result given by Theorem 6. In more detail, for lattice values a_i and a_j (i.e., values corresponding to position given by indices i and j in the table) we consider an interval $\text{Bounds}(i, j) \subseteq L$ which is

$$\text{Bounds}(i, j) = [b, a_i \wedge a_j]$$

where

$$b = \bigvee\{a_{\min(k,l)} \otimes a_{\max(k,l)} \,|\, (k = i \text{ and } a_l \prec a_j) \text{ or } (a_k \prec a_i \text{ and } l = j)\}$$

where $a_m \prec a_n$ denotes that a_m is covered by a_n, i.e. $a_m \leq a_n$ and $a_m \leq c \leq a_n$ implies $a_m = c$ or $a_n = c$. Then we go through all the values in $\text{Bounds}(i, j)$ and set them as the results of $a_i \otimes a_j$. Then we check (12)–(14) for $a_i \otimes a_j$ and the process continues as described above. We finish if we fill the whole table with values satisfying (12)–(14). Therefore, the algorithm for generating $\otimes$ can be described as a recursive procedure *generate-multiplications* which accepts two parameters: indices of the row and the column of the table in Fig. 6. The procedure uses a global variable *mult*, the purpose of which is to represent the two-dimensional table of $\otimes$.

Proof of soundness of procedure generate-multiplications `(i, j)`. Observe that if $\otimes$ (multiplication encoded by the two-dimensional array *mult*) does not satisfy (12)–(14), $\otimes$ is never stored. This follows directly from our previous observations. Hence, procedure *generate-multiplications* stores at most all possible multiplications of a given finite lattice. It remains to show that it stores exactly all of them. But this is also easy to see. Indeed, each $\otimes$ which is a multiplication satisfying adjointness with $\rightarrow$ on a residuated lattice satisfies (12)–(14), i.e. when *mult* encodes a portion of $\otimes$, *generate-multiplications* will always go to line 4 after its invocation. From lines 12–15 we can see that each value of $a_i \otimes a_j$ will be written (at some point) in $mult[i, j]$. Hence, the value of $a_1 \otimes a_1$ will be written in $mult[1, 1]$ during the first invocation of *generate-multiplications* $(1, 1)$ because $a_1 \otimes a_1 \in \text{Bounds}\,(1, 1)$. Then, *generate-multiplications* $(1, 2)$ will be invoked and $a_1 \otimes a_2$ will be written in $mult[1, 2]$ because $a_1 \otimes a_2 \in \text{Bounds}\,(1, 2)$, and so on. Line 16 resets the value of $mult[i, j]$ back to "undefined". This is for the sake of correct testing of (12)–(14) because the variable *mult* is shared among all invocations of *generate-multiplications*. The computation stops after finitely many steps. □

4.3 Removing automorphisms

In order to generate all non-isomorphic residuated lattices with the lattice part $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$, we exclude the isomorphic copies, which may arise when computing the multiplications $\otimes$ as described above, by selecting one representative. The representative is selected using the following lexicographic order. For two multiplications $\otimes_1$ and $\otimes_2$, we put $\otimes_1 <_\ell \otimes_2$ iff there exist $a_i, a_j \in L$ such that the following two conditions are both satisfied:

Procedure *generate-multiplications*(i, j)

```
    Data: indices 0 < i ≤ j < n − 1
 1  if (12)–(14) are not satisfied then
 2  |  return
 3  else
    |  // if current row is finished, start filling next row
 4  |  if j ≥ n − 1 then
 5  |  |  set i to i + 1
 6  |  |  set j to i
 7  |  end
    |  // if ⊗ is generated, store its value and terminate
 8  |  if i ≥ n − 1 then
 9  |  |  store mult
10  |  |  return
11  |  end
    |  // fill current table position with possible lattice values
12  |  foreach b in Bounds(i, j) do
13  |  |  set mult[i, j] to b
14  |  |  call generate-multiplications(i, j + 1)
15  |  end
    |  // mark current value as undefined
16  |  set mult[i, j] to "undefined"
17  end
```

(i) $k < l$ for $a_k = a_i \otimes_1 a_j$ and $a_l = a_i \otimes_2 a_j$,

(ii) $a_k \otimes_1 a_l = a_k \otimes_2 a_l$ for all $a_k, a_l \in L$ such that $k < i$ or $(k = i$ and $l < j)$.

Obviously, $<_\ell$ defines a strict total order on all possible multiplications (binary operations, in general) on $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$. Denote by $\leqslant_\ell$ the reflexive closure of $<_\ell$.

Consider now two distinct adjoint pairs $\langle \otimes_1, \to_1\rangle$ and $\langle \otimes_2, \to_2\rangle$ computed by the above backtracking algorithm and the corresponding residuated lattices $\mathbf{L}_1 = \langle L, \wedge, \vee, \otimes_1, \to_1, 0, 1\rangle$ and $\mathbf{L}_2 = \langle L, \wedge, \vee, \otimes_2, \to_2, 0, 1\rangle$. It is easily seen that $\mathbf{L}_1$ and $\mathbf{L}_2$ are isomorphic iff there is a lattice automorphism $h : L \to L$ such that $a \otimes_2 b = h(h^{-1}(a) \otimes_1 h^{-1}(b))$. Thus, we proceed as follows. After $\otimes$ is generated, we compute the set of all automorphic images $\{\otimes_i \mid i \in I\}$ of $\otimes$ and store $\otimes$ iff $\otimes$ is lexicographically least among all $\{\otimes_i \mid i \in I\}$, i.e., iff $\otimes \leqslant_\ell \otimes_i$ for all $i \in I$. Each automorphic image $\otimes_i$ of $\otimes$ is defined by $a \otimes_i b = h(h^{-1}(a) \otimes h^{-1}(b))$ where h is a lattice automorphism of $\mathbf{L} = \langle L, \wedge, \vee, 0, 1\rangle$. Therefore, in order to apply the procedure, we have to generate all automorphisms of a given finite lattice $\mathbf{L}$. This can be done in a straightforward manner using characteristic vectors and automorphism candidates introduced in Section 3.

5 Selected properties of generated structures

In this section we present basic characteristics of finite residuated lattices generated by our algorithms. We used the algorithms to generate all non-isomorphic residuated lattices with up to 12 elements. Prior to that, we generated all non-isomorphic lattices up to 12 elements. The tables summarizing the observations from this section can be found in the appendix. A database of generated lattices is available at:

`http://lattice.inf.upol.cz/order/`

Numbers of Finite (Residuated) Lattices Table 3 (see appendix) contains a basic summary. The table columns correspond to sizes of lattices (numbers of their elements). The first row contains the numbers non-isomorphic lattices. These numbers agree with observations concerning the numbers of lattices from [22]. The second row contains the numbers of non-isomorphic residuated lattices. The third row contains the numbers of non-isomorphic linearly ordered residuated lattices (i.e., lattices with every pair of elements comparable). We can see from the table that small residuated lattices tend to be linear: for $|L| = 5$, 22 residuated lattices out of 26 are linear. With growing sizes of $|L|$, the portion of linear residuated lattices decreases: for $|L| = 11$, one

fifth of all the residuated lattices are linear; for $|L| = 12$ only one seventh of all the residuated lattices are linear. Another observation concerns the relationship between (numbers of) residuated lattices and (numbers of) their distinct lattice reducts. Recall that if $\mathbf{L} = \langle L, \wedge, \vee, \otimes, \to, 0, 1\rangle$ is a residuated lattice, its reduct $\langle L, \wedge, \vee, 0, 1\rangle$ is a lattice. Thus, we may ask how many n-element lattices are reducts of n-element residuated lattices. This is shown in the last row of Table 3 which contains the numbers of pairwise distinct non-isomorphic lattice reducts of all non-isomorphic residuated lattices. For instance, the values in column corresponding to $|L| = 12$ mean: there are 262776 non-isomorphic lattices but only 38165 of them can be equipped with $\otimes$ and $\to$ to form a residuated lattice. Notice that even if the number of residuated lattices rapidly grows with growing $|L|$, the number of their lattice reducts compared to the number of all lattices (of that size) decreases. This means that with growing $|L|$, the average number of residuated lattices with the same lattice part increases. For instance, for $|L| = 8$ the average number of residuated lattices sharing the same lattice part is approximately 77 while for $|L| = 12$ it is 803.

Heights and Widths of Finite (Residuated) Lattices The values in Table 3 may suggest that most residuated lattices can be found on n-element chains. This is so for smaller residuated lattices but it is no longer true for larger lattices. Namely, consider the heights and widths of the lattices. A *height* (or *width*) of a lattice is the length of the longest maximal chain (or antichain) contained in that lattice. For instance, for $|L| = 12$, we can depict the numbers of lattices according to their width and height as in Table 4 (see appendix). The rows and columns in Table 4 represent heights and widths of lattices, respectively. The table entries represent the numbers of non-isomorphic lattices with the dimensions given by the corresponding rows and columns. In a similar way, we can depict the numbers of distinct residuated lattices according to their width and height as in Table 5. Table 5 shows that the lattice parts of most residuated lattices are "high and thin" but in case of $|L| = 12$, the most frequent residuated lattices are those with width 2 (second column of Table 5). Let us mention that the distribution of all lattices and all lattice reducts according to their dimensions is quite different from that of residuated lattices. Indeed, the distributions of lattices in Table 4 and Table 6 are similar but quite different from that in Table 5. Analogous observations can be made for all generated finite residuated lattices and their lattice reducts with $|L| < 12$.

Numbers of Finite (Residuated) Lattices Satisfying Additional Conditions Table 7 provides a summary of the numbers of non-isomorphic lattices which are modular, distributive, have complements, are Boolean, have relative complements, pseudo-complements, and relative pseudo-complements [19]. Note that the lines for all, modular, and distributive lattices are known [34], and that it is also know that the numbers in line 3 and line 8 coincide [7]. In addition to that, we consider the following properties of residuated lattices (see [4, 12, 19, 20]):

(MOD)	$a \leq c$ implies $a \vee (b \wedge c) = (a \vee b) \wedge c$	(*modularity*)
(DIS)	$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$	(*distributivity*)
(MTL)	$(a \to b) \vee (b \to a) = 1$	(*prelinearity*)
(Π1)	$(c \to 0) \to 0 \leq ((a \otimes c) \to (b \otimes c)) \to (a \to b)$	(Π1-*property*)
(Π2)	$a \wedge (a \to 0) = 0$	(Π2-*property*)
(STR)	$(a \otimes b) \to 0 = (a \to 0) \vee (b \to 0)$	(*strictness*)
(WNM)	$((a \otimes b) \to 0) \vee ((a \wedge b) \to (a \otimes b)) = 1$	(*weak nilpotent minimum*)
(DIV)	$a \wedge b = a \otimes (a \to b)$	(*divisibility*)
(INV)	$a = (a \to 0) \to 0$	(*involution*)
(IDM)	$a = a \otimes a$	(*idempotency*)

These properties are of interest, e.g., when lattices are considered as structures of truth values in many-valued logics and fuzzy logics [4, 20]. Table 8 contains the numbers of residuated lattices satisfying these conditions. Table 9 summarizes the numbers of algebras (particular residuated lattices) which are defined by a combination of the above-mentioned properties. The tables show that BL-algebras are very rare among residuated lattices up to 12 elements. The situation for MTL-algebras is better but still, only 15 % of all 12-element residuated lattices are MTL-algebras. An observation which may be surprising is that (Π1) is far more frequent a property than prelinearity (for $|L| \leq 12$).

Relationship Between Properties of Finite (Residuated) Lattices Table 7 shows the numbers of lattices having each property but does not show, e.g., how many modular lattices are pseudo-complemented; similarly for Table 8 and Table 9. To reveal dependencies among properties of lattices and residuated lattices,

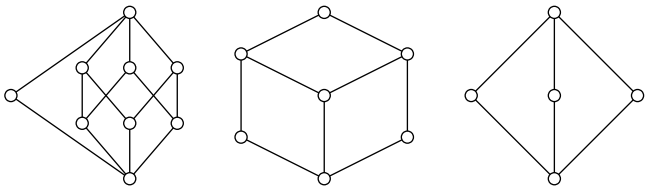


Figure 7: Least lattices that have a specific group of properties.

we constructed Table 10, Table 11, and Table 12. These tables show all combinations of lattice and residuated lattice properties which appear in the generated databases. In case of Table 10, the columns denote the same properties of lattices considered in Table 7. The left-most column contains numbers of non-isomorphic lattices with given combinations of properties. Each row of the tables represents one combination of properties (properties which are present are marked by "×"). From Table 10 we can see that some combinations of properties are rare. In addition to that, some combinations of properties do not appear in "small" lattices (up to certain number of elements). For instance, the least lattice which is only relatively complemented and (in consequence) complemented has 9 elements and it is depicted in Fig. 7 (left). The least lattice which does not satisfy any of the properties MOD–RPC (see Table 10) has 7 elements and is depicted in Fig. 7 (middle). The lattice in Fig. 7 (right) is the least lattice which is only modular, complemented, and relatively complemented. Table 11 and Table 12 depict dependencies among properties of the generated residuated lattices. Again, some combinations of properties are rare and some of them appear only in larger structures. For illustration, the least residuated lattice which satisfies only (MOD) and (Π2) has 9 elements.

Acknowledgment

Partly supported by grant P202/10/0262 of the Czech Science Foundation and by institutional support, research plan MSM 6198959214. The paper is an extended version of a contribution presented at IFSA 2007 Congress.

References

[1] Aït-Kaci H., Boyer R., Lincoln P., Nasr R.: Efficient implementation of lattice operations. *ACM Trans. Program. Lang. Syst.* **11**(1)(1989), 115–146.

[2] Bartušek T., Navara M.: Program for generating fuzzy logical operations and its use in mathematical proofs. *Kybernetika* **38**(3)(2002), 235–244.

[3] Birkhoff G.: *Lattice Theory.* Publ. AMS, Providence, RI (1967).

[4] Belohlavek R.: *Fuzzy Relational Systems: Foundations and Principles.* Kluwer Academic/Plenum Publishers, New York, 2002.

[5] Blount K., Tsinakis C.: The structure of residuated lattices. *Intl. Journal of Algebra and Computation* **13**(2003), 437–461.

[6] Carpineto C., Romano G.: *Concept Data Analysis. Theory and Applications.* J. Wiley, 2004.

[7] Chajda I., Glazek K.: *A Basic Course on General Algebra*, Technical University Press, Zielona Góra, 2000.

[8] De Baets B.: The number of linearly ordered abelian integral monoids of size n; the number of t-norms on an n-chain. The On-Line Encyclopedia of Integer Sequences (database entry A030453).

[9] De Baets B., Mesiar R.: Triangular norms on product lattices. *Fuzzy Sets and Systems* **104**(1999), 61–75.

[10] Dilworth R. P.: Abstract residuation over lattices. *Bull. Amer. Math. Soc.* **44** (1938), 262–268.

[11] Erne M., Stege K.: Counting finite posets and topologies. *Order* **8**(1991), 247–265.

[12] 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.

[13] Freese R. S., Ježek J., Nation J. B.: *Free Lattices.* Math. Surveys and Monographs, Vol. 42, AMS, 1995.

[14] Ganter B., Wille R.: *Formal Concept Analysis. Mathematical Foundations.* Springer, Berlin, 1999.

[15] Galatos N., Jipsen P., Kowalski T., Ono H.: *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*, Volume 151 (Studies in Logic and the Foundations of Mathematics), Elsevier, 2007.

[16] Goguen J. A.: L-fuzzy sets. *J. Math. Anal. Appl.* **18**(1967), 145–174.

[17] Goguen J. A.: The logic of inexact concepts. *Synthese* **18**(1968-9), 325–373.

[18] Gottwald S.: *A Treatise on Many-Valued Logics.* Research Studies Press, Beldock, England, 2001.

[19] Grätzer G. A.: *General Lattice Theory.* Birkhauser (1998, 2nd ed.).

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

[21] Heitzig J., Reinhold J.: The number of unlabeled orders on fourteen elements. *Order* **17**(4)(2000), 333–341.

[22] Heitzig J., Reinhold J.: Counting finite lattices. *Algebra Universalis* **48**(1)(2002), 43–53.

[23] Höhle U.: On the fundamentals of fuzzy set theory. *J. Math. Anal. Appl.* **201**(1996), 786–826.

[24] Jipsen P.: Small residuated lattices. `http://www1.chapman.edu/~jipsen/gap/rl.html`.

[25] Jipsen P., Tsinakis C.: A survey of residuated lattices. In: J. Martinez (Ed.): *Ordered Algebraic Structures*, Kluwer Academic Publishers, Dordrecht, 2002, 19–56.

[26] Kleitman D. J., Rothschild B. L.: Asymptotic enumeration of partial orders on a finite set. *Trans. Amer. Math. Soc.* **205**(1975), 205–220.

[27] Kleitman D. J., Winston K. J.: The asymptotic number of lattices. *Ann. Discrete Math.* **6**(1980), 243–249.

[28] Klement E. P., Mesiar R., Pap E.: *Triangular Norms.* Kluwer, 2000.

[29] Klir G. J., Yuan B.: *Fuzzy Sets and Fuzzy Logic. Theory and Applications.* Prentice Hall, 1995.

[30] Kyuno S.: An inductive algorithm to construct finite lattices. *Math. of Computation* **33**(145)(1979), 409–421.

[31] Lygeros N., Zimmermann P.: Computation of $P(14)$, the number of posets with 14 elements: 1.338.193.159.771. `http://www.lygeros.org/Math/poset.html`.

[32] Miller G. T.: The magical number seven, plus or minus two: some limits on our capacity for processing information. *The Psychological Review* **63**(1956), 81–97.

[33] Pavelka J.: On fuzzy logic I, II, III. *Z. Math. Logik Grundlagen Math.* **25**(1979), 45–52, 119–134, 447–464.

[34] The On-Line Encyclopedia of Integer Sequences, `http://www.research.att.com/~njas/sequences/`.

[35] Ward M., Dilworth R. P.: Residuated lattices, *Trans. Amer. Math. Soc.* **45** (1939), 335–354.

	1	2	3	4	5	6	7	8	9	10	11	12
lattices	1	1	1	2	5	15	53	222	1078	5994	37622	262776
residuated lattices	1	1	2	7	26	129	723	4712	34698	290565	2779183	30653419
linear res. lattices	1	1	2	6	22	94	451	2386	13775	86417	590489	4446029
res. lattice reducts	1	1	1	2	3	7	18	61	239	1125	6138	38165

Table 3: Numbers of non-isomorphic finite lattices and residuated lattices up to 12 elements.

	1	2	3	4	5	6	7	8	9	10
3										1
4					99	395	288	98	17	
5				3847	14418	9536	2115	176		
6			3531	37813	43394	12050	952			
7		87	15501	48261	23595	2507				
8		666	14735	17380	3117					
9		849	4704	1792						
10		350	456							
11		45								
12	1									

Table 4: Numbers of 12-element lattices with given heights and widths.

	1	2	3	4	5	6	7	8	9
4									1
5				3	127	165	88	48	
6			240	9383	22627	9638	1335		
7		236	99088	332299	161275	18546			
8		121970	1363290	1009364	142551				
9		1732870	3563657	733266					
10		6007716	2709365						
11		8168242							
12	4446029								

Table 5: Numbers of 12-element residuated lattices with given heights and widths.

	1	2	3	4	5	6	7	8	9
4									1
5				2	123	159	72	15	
6			92	2215	3295	1139	126		
7		11	2362	8498	4397	518			
8		241	4549	5377	973				
9		455	2183	805					
10		239	280						
11		37							
12	1								

Table 6: Numbers of 12-element lattice reducts with given heights and widths.

	1	2	3	4	5	6	7	8	9	10	11	12
all lattices	1	1	1	2	5	15	53	222	1078	5994	37622	262776
modular	1	1	1	2	4	8	16	34	72	157	343	766
distributive	1	1	1	2	3	5	8	15	26	47	82	151
complemented	1	1	0	1	2	6	18	71	307	1594	9446	63461
Boolean	1	1	0	1	0	0	0	1	0	0	0	0
relatively complemented	1	1	0	1	1	1	1	2	2	4	6	13
pseudo-complemented	1	1	1	2	4	10	29	99	391	1775	9214	54151
relatively pseudo-complemented	1	1	1	2	3	5	8	15	26	47	82	151

Table 7: Numbers of non-isomorphic lattices with selected properties.

	1	2	3	4	5	6	7	8	9	10	11	12
all res. lattices	1	1	2	7	26	129	723	4712	34698	290565	2779183	30653419
modular	1	1	2	7	26	125	660	3923	25445	180113	1389782	11798582
distributive	1	1	2	7	26	124	645	3792	24268	169553	1290956	10823436
(Π1) identity	1	1	1	4	9	46	240	1610	12679	118052	1280764	16074272
prelinear	1	1	2	7	23	99	464	2453	14087	88188	601205	4516962
(Π2) identity	1	1	1	3	8	30	143	794	5090	37036	306456	2897889
strict	1	1	1	3	7	27	129	726	4713	34705	290565	2779212
(WNM) identity	1	1	2	5	11	30	78	238	771	2908	12812	67467
divisible	1	1	2	5	10	23	49	111	244	545	1203	2697
involutive	1	1	1	3	3	12	15	70	112	493	980	4325
idempotent	1	1	1	2	3	5	8	15	26	47	82	151

Table 8: Numbers of residuated lattices with selected properties.

	1	2	3	4	5	6	7	8	9	10	11	12
all res. lattices	1	1	2	7	26	129	723	4712	34698	290565	2779183	30653419
MTL-algebras	1	1	2	7	23	99	464	2453	14087	88188	601205	4516962
SMTL-algebras	1	1	1	3	7	24	99	467	2454	14094	88188	601231
WNM-algebras	1	1	2	5	9	21	40	90	180	378	757	1584
BL-algebras	1	1	2	5	9	20	38	81	160	326	643	1314
SBL-algebras	1	1	1	3	5	10	20	41	82	165	326	655
IMTL-algebras	1	1	1	3	3	8	12	35	61	167	333	971
Heyting algebras	1	1	1	2	3	5	8	15	26	47	82	151
G-algebras	1	1	1	2	2	3	3	5	6	8	8	12
NM-algebras	1	1	1	2	1	2	1	4	3	3	2	6
MV-algebras	1	1	1	2	1	2	1	3	2	2	1	4
Π-algebras	1	1	0	1	0	0	0	1	0	0	0	0
ΠMTL-algebras	1	1	0	1	0	0	0	1	0	0	0	0

Table 9: Numbers of selected algebras (particular residuated lattices).

	MOD	DIS	COM	BOO	REL	PCO	RPC
168660							
72930			×				
62811						×	
1945			×			×	
580	×					×	
473	×						
338	×	×				×	×
19			×		×		
10	×		×		×		
4	×	×	×	×	×	×	×

Table 10: Numbers of lattices sharing selected properties. (legend: MOD = modular, DIS = distributive, COM = complemented, BOO = boolean, REL = relatively complemented, PCO = pseudo-complemented, RPC = relatively pseudo-complemented).

	MTL-algebra	SMTL-algebra	WNM-algebra	BL-algebra	SBL-algebra	IMTL-algebra	G-algebra	NM-algebra	MV-algebra	Π-algebra	ΠMTL-algebra
28539974											
4511103	×										
705260	×	×									
2954	×		×								
1556	×					×					
1258	×	×		×	×						
1234	×			×							
48	×	×	×	×	×		×				
39	×		×	×							
19	×		×			×		×			
13	×			×		×			×		
4	×		×	×		×		×	×		
4	×	×	×	×	×	×	×	×	×	×	×

Table 11: Numbers of residuated lattices sharing selected properties.

	MOD	DIS	MTL	Π1	Π2	STR	WEA	DIV	INV	IDE
12275528				×						
6404789										
3252773	×	×								
3025931	×	×	×							
3009686	×	×		×						
1509962					×	×				
1485172	×	×	×	×						
781939	×	×			×	×				
705260	×	×	×		×	×				
653138	×			×						
317172	×									
110710	×				×	×				
101016					×					
55732				×			×			
34162	×	×			×					
15501							×			
5761	×	×					×			
2713	×	×	×				×			
2271				×					×	
1720	×	×		×					×	
1719	×						×			

	MOD	DIS	MTL	Π1	Π2	STR	WEA	DIV	INV	IDE
1556	×	×	×	×					×	
1509	×				×					
1258	×	×	×		×	×		×		
1234	×	×	×					×		
1189	×			×			×			
977	×	×		×			×			
757	×	×			×	×		×		
630	×	×						×		
537	×	×			×			×		
419	×			×					×	
241	×	×	×	×			×			
152	×	×			×	×	×	×		×
138	×	×			×		×	×		×
77	×	×					×	×		
48	×	×	×		×	×	×	×		×
39	×	×	×				×	×		
19	×	×	×	×			×		×	
13	×	×	×	×				×	×	
10	×	×		×			×		×	
4	×	×	×	×			×	×	×	
4	×	×	×	×	×	×	×	×	×	×

Table 12: Numbers of residuated lattices sharing selected properties (detail).