Pruning Techniques in Łukasiewicz Logics

Raed Basbous

Al-Quds Open University, Faculty of Technology and Applied Sciences,
P6160655 Ramallah, P.O. Box 58100, Palestine
E-mail: rbasbous@qou.edu

Benedek Nagy

Eastern Mediterranean University, Faculty of Arts and Sciences, Department of Mathematics, Famagusta, North Cyprus, via Mersin 10, Turkey
Eszterházy Károly Catholic University, Faculty of Informatics, Department of Computer Science, 3300 Eger, Leányka út 4/A, Hungary
E-mail: benedek.nagy@emu.edu.tr

Tibor Tajti

Eszterházy Károly Catholic University, Faculty of Informatics, Department of Computer Science, 3300 Eger, Leányka út 4/A, Hungary
University of Debrecen, Faculty of Informatics, Department of Information Technology, 4028 Debrecen, Kassai út 26, Hungary
E-mail: tajti.tibor@uni-eszterhazy.hu

Abstract: Short circuit evaluations and related pruning techniques play important roles in logic, artificial intelligence and computer science, especially, in software engineering, decision-making and hardware design. Here, we consider a well-known and widely applied fuzzy logic system, the Łukasiewicz logic. We present and prove various pruning techniques to make the evaluations of logical formulas faster and more efficient, by cutting those branches of the formula tree that have no influence on the result at the root. Complex examples show the efficiency of our pruning techniques, which are used to do the evaluation of formulas faster in Łukasiewicz logic. This logic is appropriate for various engineering applications related to fuzzy technology and decision making; therefore, our results are important for fast evaluation.

Keywords: many valued and fuzzy logics; fuzzy decision; generalized alpha-beta pruning; short-circuit evaluation; cuts off of expression trees

1 Introduction

Pruning techniques are well known in various places in science, technology and engineering, to accelerate the evaluation of some formulas or decision trees. This can be done in cases when the result is known with 100% certainty, without evaluating the subresults of each part of the computation. Under the name "short circuit evaluations" pruning techniques appear in programming languages, for instance, in C (see, e.g. [1]). These methods are based on well-known logical laws of propositional logic. This classical logic, also known as Boolean logic, is widely known and can be read from various sources, including articles and books on software engineering, (discrete) mathematics, computer sciences, hardware design, philosophical and mathematical logics, linguistics, etc., since Boolean logic is known as one of the foundations of electric engineering, computer sciences, mathematics, information technology, and other practical or research areas [2]. In addition to Boolean logic, several additional branches have been developed, for many different purposes. In predicate logic, quantifiers appear; with their help more advanced formulas can be used. In temporal and modal logics, new operations were introduced, e.g., operations involving the time dependence of truth and, for example, necessity and possibility, respectively. In many valued and fuzzy logics, another feature is extended: the used truth values are no more limited to only the classical crisp false and true values. In classical logic, there are many paradoxes. One may resolve some by using many-valued or fuzzy logics [3] [4]. The liar paradox is a very old and famous example [5]. The sentence "This sentence has a false value." can be neither true nor false; and we do not have other possible truth value(s) in Boolean logic. The existence of at least one more truth value could lead to the solution of this paradox. Starting from the 1920s numerous fuzzy logics were investigated. The most known such fuzzy logic systems include the Gödel type logic, the Łukasiewicz type logic, and the product logic [4, 6-9]. Gödel logic is special in the sense that the law of double negation does not function as a logical law. [9]. The Gödel logic is modelling an optimistic world, for example, to achieve maximum profit in a cooperative environment. Product logic can be a good choice for modelling a realistic and tolerant environment with various sovereign partners. Our main focus, the Łukasiewicz logic can be considered in a pessimistic, unfriendly environment where partners aim for minimal losses in competition. [4]. Deductions in many-valued logics are translated to mixed integer programming problems in [10]. There are various applications of fuzzy logics and fuzzy sets [11] [12]. The connection between Łukasiewicz type logic and fuzzy sets is discussed, e.g., in [3], moreover, Łukasiewicz type predicate logic was also developed in the past decades to allow to use the nice properties of advantages of Łukasiewicz logic in first-order logics, see, e.g. [13] [14].

Logical expressions can be found everywhere, engineering and programming cannot be done without them. Therefore, their evaluation is an important task both in theory and practice. In this paper, a very popular and well-known fuzzy logic, the Łukasiewicz type logic is considered with infinitely many truth-values. First, some

preliminaries are recalled (see Section 2) including a brief description of Łukasiewicz logic and short circuit evaluation techniques used in Boolean logic, artificial intelligence, and game theory. Then, in Section 3, various pruning techniques are proven for the expressions in Łukasiewicz logic. Complex examples are also provided in Section 4 showing the efficiency (reduced size of the expression trees, much faster evaluations) of the proposed techniques. Finally, Section 5 presents the conclusions for this work.

2 Preliminaries

We start the section by recalling the concepts of expression trees. Some notions of Boolean logic and some pruning techniques including the short circuit evaluation used in classical logic are also recalled. Finally, a brief description of Łukasiewicz logic is also given.

2.1 Tree Representation of Expressions

Expressions are used for formal description of mathematics, logic, and various sciences. There are simple expressions and various connectives/operators are used to build complex expressions. These connectives are usually unary and binary ones; thus, the expression can be drawn as a binary tree. The main connective is in the root of the tree, while other connectives are in the other non-leaf nodes. The leaf nodes represent those simple expressions that we used to build our complex expressions. Nodes representing unary connective have exactly one child, while nodes of binary connectives have two children. Every node of the tree represents a subformula that is represented by the maximal subtree rooted in the given node. By definition, the expression is evaluated in a bottom-up manner. We start with the leaves, their values are used to evaluate the subformulas represented by their parents, etc. When all subformulas, i.e., all nodes, but the root are evaluated, in the final step the result for the whole formula is obtained.

2.2 Boolean Logic

The two-valued classical propositional (crisp) logic is widely known and used in theoretical and applied sciences. It was formalized by Boole in the XIX century; hence it is also called Boolean logic (and Boolean algebra). There are two (truth) values: they are interpreted as true (T, 1) and false (F, 0). Since this logic is used in electrical switching circuits, it serves as the foundation for all of our digital machines, including digital calculators or computing devices. The textbook [2] is recommended for those who are unfamiliar with classical logic.

The syntax of propositional logic is as follows. There are infinitely many propositional (also called Boolean) variables. They, together with T and F are the atomic formulas. As usual in Boolean logic, we use the conjunction ("logical and", shortly L-and), disjunction ("logical or", L-or), implication, and negation operators. This latter operator is unary, all others are binary. If A and B are two logical formulas (maybe atomic, maybe not), then their L-and ($A \wedge B$), L-or ($A \vee B$), and implication ($A \rightarrow B$) are also logical formulas. The formulas A and B are referred to as the original formula's main subformulas. The negation $\neg A$ of a logical formula A is also a logical formula. Finally, all logical formulas are made up from atomic formulas by applying finitely many connectives. A formula tree can be used to represent the logical formula.

The semantic rules can be seen in Table 1; with their help any logical formula can be evaluated if the truth-values of the present propositional variables are fixed. T has the value 1, while F has the value 0, variables may have either value.

In Boolean logic, the value of the formula is always 0 or 1. However, there are various cases where the truth-value of one of the two main subformulas is sufficient to determine the value of the main formula. Therefore, in many cases, we may not need a full evaluation, but short circuit evaluation works: some nodes in the formula tree can be omitted because the final value of the formula is not influenced by their values. In the next subsection, we start with the two trivial cases of these shortcut techniques.

Table 1
The semantics of classical logic. (The first columns show the possible values of variable A; the possible values of variable B are shown in the first row at binary operators.)

A	$\neg$A
0	1
1	0

A$\wedge$B	0	1
0	0	0
1	0	1

A$\vee$B	0	1
0	0	1
1	1	1

A$\rightarrow$B	0	1
0	1	1
1	0	1

2.3 Short Circuit Evaluations and other Cut off Methods

Short circuit evaluation is used for various purposes: it saves time, but it is also used because of safety considerations [15]. In the C programming language, the symbols && and || mean "L-and" and "L-or", respectively. By evaluating these operations short circuit is applied as follows:

At L-and, if one of the conditions/arguments is already known to be false, then value 0 can be assigned to the L-and node without checking the other child(ren).

At L-or, if it is known that one of the conditions/arguments is true, then the L-or node has the value 1 without checking the value of the other child(ren).

Pruning techniques are also frequently used in artificial intelligence and game theory when a decision is computed based on a game tree (or its part). These methods are called alpha-beta pruning, see, e.g., [15] [16]. Game trees, theoretically,

that are comprising all possible instances (matches) and outcomes, are used to represent combinatorial, two-player, zero-sum, full information, finite, and deterministic games with possible moves of the players. The minimax algorithm gives the value (solution) of the game by evaluating the game tree, and subsequently, it provides the optimal strategies and best-guaranteed payoffs for both players. The alpha-beta pruning, similar to the short circuit evaluation methods, cuts off some branches of the game tree that has no influence on the final result. Alpha-beta pruning works on trees where the values are not necessarily restricted to be in {0,1}. Figure 1 shows an example in a small game tree, for every vertex in a tree when α becomes greater than or equal to β, we can stop expanding its children. These algorithms are presented in detail in [15-18].

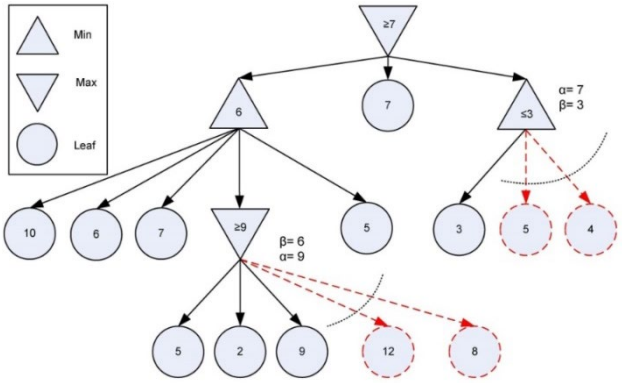

Figure 1
An example of alpha-beta pruning

2.4 Łukasiewicz Logics

Among others, Łukasiewicz was pioneering to extend the classical logic by introducing intermediate truth-values violating Aristotle's law of excluded middle. Łukasiewicz defined logics for arbitrary many ($n \geq 2$) and even for infinitely many truth values [7, 8, 19]. His infinite-valued logic is still one of the most attractive candidates of fuzzy logic [20]. Every real number of the closed interval [0,1] is a possible truth-value. The logical connectives connected to his logic are the Łukasiewicz implication ($\rightarrow$), the negation ($\neg$), the Łukasiewicz conjunction ($\otimes$), and the Łukasiewicz disjunction ($\oplus$). The syntax of his logic is exactly the same as the syntax of the classical logic as we have already described, but every element of [0,1] plays the role of a constant (not only T and F, i.e., 1 and 0).

The semantics of Łukasiewicz logic are as follows. The variables can have values from the real interval [0,1] inclusive of the two classical values, also each element of [0,1] could play the role of a constant. The truth-values of complex formulas are calculated from the value of their main subformulas according to the main operator [8, 19, 20] as shown in equations (1)-(4):

$$|\neg A| = 1 - |A| \tag{1}$$

$$|A \rightarrow B| = \begin{cases} 1 - |A| + |B|, \text{if } |A| > |B| \\ 1, \text{otherwise} \end{cases} \tag{2}$$

$$|A \otimes B| = \begin{cases} |A| + |B| - 1, \text{if } |A| + |B| > 1 \\ 0, \text{otherwise} \end{cases} \tag{3}$$

$$|A \oplus B| = \begin{cases} |A| + |B|, \text{if } 1 > |A| + |B| \\ 1, \text{otherwise} \end{cases} \tag{4}$$

When a formula is evaluated, the leaf nodes of the expression trees have values from the [0,1] interval. Again, the (truth) value of the main formula (i.e., the tree) is computed by the bottom-up strategy. As in the case of classical logic, some branches (some subformula) could be needless to compute. This is the main task of this paper. Before turning to the pruning techniques, we note that the finite k-valued variants (where $k > 1$ is any integer) of Łukasiewicz logic work with the values $0 = \frac{0}{k-1}, \frac{1}{k-1}, \ldots, \frac{k-1}{k-1} = 1$ Case $k = 2$ is exactly the classical logic. The conjunction and disjunction in the Łukasiewicz logic are also called "bounded product" and "bounded sum", respectively.

3 Pruning Techniques For Formula Trees In Łukasiewicz Logic

This section proposes several techniques for improving the evaluation of expression trees in Łukasiewicz logic using the aforementioned logical connectives. Thus, we deal with trees with a bounded set of truth values: the real numbers from the [0,1] interval can be used at the tree's vertices. They are given at the tree's leaves at the start of the evaluation, and the task is to calculate the value for the root node. The restriction in the layout of these trees is that vertices with negation must have exactly one child, whereas vertices assigned to any other connectives must have two children, referred to as left and right child, respectively.

In the next subsections, we describe in detail the proposed pruning techniques to accelerate the evaluation method which can be used for such trees.

3.1 Disjunction (⊕)and Conjunction (⊗) Pruning

Evaluation of L-or (⊕) and L-and (⊗) nodes may be accelerated in various ways, which depend on the left and right children of these nodes. Although the next two results are evident expansions of the well-known short circuit evaluation techniques mentioned earlier, for the sake of completeness, we present them here.

Theorem 1. For a conjunction vertex Γ, a shortcut can be applied if it has a child having value 0. The value 0 can be assigned to Γ independently of the value of the vertex that is connected to the other side.

Proof. Knowing that one of the children is evaluated to 0, it follows that the result of equation (3) equals to 0 (the minimum value) whatever the value of the vertex that is the child on the other side of Γ. This is based on the logical law A ⊗ 0 has value 0. □

Remark 1. The evaluation of a node could go faster if it starts by getting the value of a leaf child if it exists.

Figure 2 shows an example of such a case where this cut can be applied. We have an analogous result about nodes associated with a disjunction.

Theorem 2. For a disjunction vertex Δ, if one of its children is evaluated to 1, then a cut can be applied assigning value 1 to Δ independently of the value of the other child.

Proof. A child (maybe a leaf) with value 1 makes the result of equation (4) equal to 1 (possible maximum value), based on the law: A ⊕ 1 is always 1. □

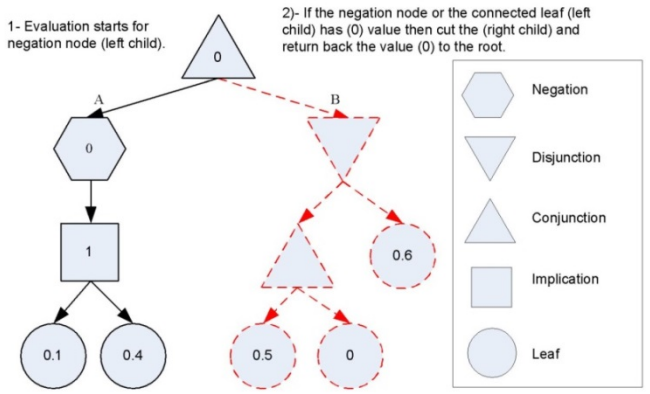

Figure 2

A cut is used at a L-and (⊗) vertex if one child (the left or right) has a value of 0. Only 3 operator nodes plus 2 leaves (altogether of 5 nodes) are explored and evaluated out of 5 operator nodes plus 5 leaves (altogether of 10 nodes) to obtain the final value at the root.

To optimize the evaluation, the same process can be used here as for the L-and vertices. For an example see Figure 3.

Beyond the preceding techniques which can be applied when the minimal/ maximal value is reached, a more advanced method can also be effective with respect to conjunction and disjunction nodes. In these techniques we go deeper in the formula tree and visit some grandchildren of the chosen node as well.

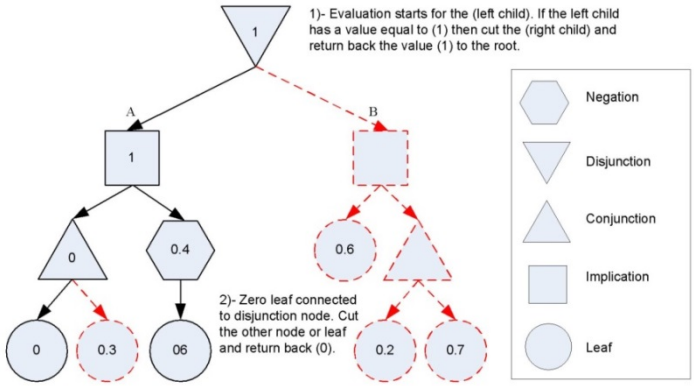

Figure 3

A cut is used when a L-or (⊕) vertex has one child with a truth value 1. Only 4 operator nodes plus 2 leaves (a total of 6 vertices) are explored and evaluated from the 6 operator nodes plus 6 vertices which are leaves (a total of 12 vertices) to obtain the final value in the root.

Theorem 3. Let a disjunction node Δ have both children (X_1 and X_2) among the following types, i.e., $X_i \in \{\Phi_i, \Pi_i, \Delta_i\}$ (i=1,2), where:

- Φ_i is a negation vertex with a conjunction child Γ_i
- Π_i is an implication vertex
- Δ_i is a disjunction vertex

Let Y_1 and Y_2 denote the children of the vertex Γ_1, Π_1 or Δ_1 depending on the cases above; and let Y_3 and Y_4 denote the children of the vertex Γ_2, Π_2 or Δ_2. Then, after knowing some of the values of the vertices Y_i we may apply a cut. For vertex Δ_1 (Δ_2) let their children's value be denoted by y_1 and y_2 (y_3 and y_4, respectively). In the case of vertex Π_1 (Π_2) let y_2 (y_4, respectively) denote the value of its right child and let y_1 (y_3, respectively) denote the difference of 1 and the value of its left child. For vertex Γ_1 (Γ_2) let denote y_1 and y_2 (y_3 and y_4, respectively) the difference of 1 and its children's values. Then, at any phase of the evaluation, if the sum of the already evaluated values y_i is at least 1, a cut can be used to assign the value 1 to Δ, independently of the values of the remaining (not yet evaluated) vertices Y_i.

Proof. The proposed method is based on equations (2), (3) and (4) depending on the connectives at the children nodes. In the case of disjunction child(ren), by equation (4), we can see that its value can be at most 1, and the expression has a final value which is always larger or equal to both |A| and |B|. When the sum of the values of the given successors (e.g., A and B in this case) is larger or equal to 1, the disjunction has its maximal value, 1. In case of implication child(ren), Π_1 and/or Π_2, observing equation (2), the value is always at least the value of the second (i.e., right) child: $| A \to B | \geq | B |$. Thus, for Y_i that is a child of a disjunction or right child of an implication, its own value is used in y_i.

Obviously, the result of the expression in (2) cannot be less than the negation of its left child (Y_1 and/or Y_3), let us say, represented by formula A, which is equal to $1 - |A|$. In the case of conjunction descendants (Γ_1 and/or Γ_2), their minimum value, by the expression in (3), i.e., 0, is obtained when the sum of the values of the two connected successors (let us say, formulas A and B) is less than 1; and the value of the expression cannot be larger than |A| and also than |B|. However, here we use their negations, thus, the value of the corresponding vertex $X_i = \Phi_i$ is $1 - \max(|A| + |B| - 1, 0) = \min(1 - (|A| + |B| - 1), 1 - 0) = \min((1 - |A|) + (1 - |B|)), 1)$. Hence, we use the difference of 1 and the children's values in these cases. When some of the values in y_i are already known, and their sum is already at least 1, then the value of Δ cannot be less than 1. □

We note here that the technique described above can be used in many ways in practice, e.g., by evaluating the left successors of both X_1 and X_2 first, as we explain in the following examples. Here, the evaluation of the connected vertices can be started in parallel (e.g., by the left child of both of these vertices), starting by getting the value of the connected leaf if such a child exists. After evaluating or getting the value of the first (left) successors of both connected children vertices, we may make a cut-off, as it is shown in the examples of Figure 4 having L-or at both children and Figure 5 having an implication and an L-or as children.

Now we show an analogous theorem for conjunction vertices.

Theorem 4. Let a conjunction node Γ have both children (X_1 and X_2) among the following types, i.e., $X_i \in \{\Phi_i, \Gamma_i\}$ (i=1,2), where:

- Φ_i is a negation vertex, with disjunction child Δ_i or implication child Π_i
- Γ_i is a conjunction vertex

Let Y_1 and Y_2 denote the children of the vertex Γ_1 (Δ_1 or Π_1, resp.) depending on the cases above; and let Y_3 and Y_4 denote the children of the vertex Γ_2 (Δ_2 or Π_2). For vertex Γ_1 (Γ_2) let their children's value be denoted by y_1 and y_2 (y_3 and y_4, respectively). In the case of vertex Δ_1 (Δ_2), let denote y_1 and y_2 (y_3 and y_4, respectively) the difference of 1 and its children's values. In the case of Π_1 (Π_2), let denote y_1 (y_3) the value of its left child and y_2 (y_4) the difference between 1 and the value of its right child. Then, at any phase of the evaluation, if the sum of the already evaluated values y_i is at most their number – 1, then a cut can be applied assigning value 0 to Γ, independently of the values of the remaining (not yet evaluated) vertices Y_i.

Proof. In this theorem, a cut is made by obtaining the minimal value at a conjunction vertex Γ similar to in Theorem 1. However, here we go more deeply into the formula tree. In the case of conjunction child(ren) Γ_i the values

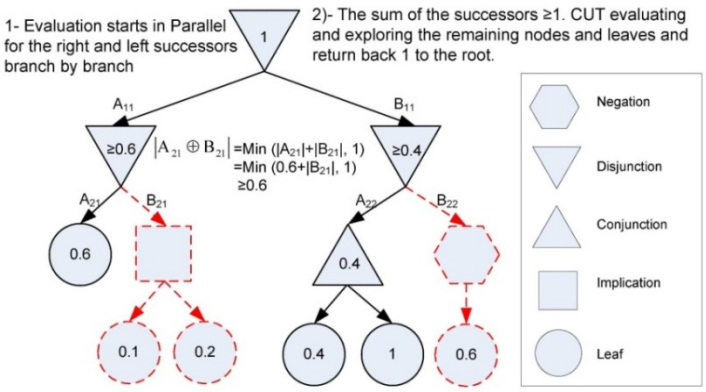

Figure 4

A pruning example at an L-or node that has two L-or children and their sum is not less than 1. Only 4 operator nodes plus 3 leaves (altogether 7 nodes) out of 6 operator nodes plus 6 leaves (altogether 12 nodes) are checked and evaluated in the entire computation.

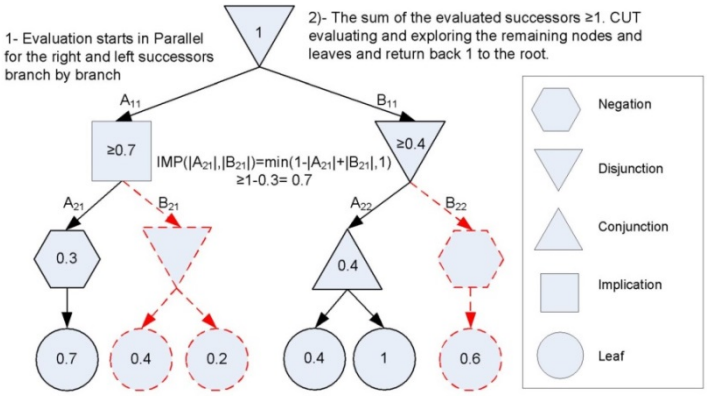

Figure 5

A pruning example applied at an L-or node having an implication node (left child) and an L-or node (right child) such that their sum is larger than or equal to 1. 5 operator nodes plus 3 leaves (altogether 8 nodes) out of 7 operator nodes plus 6 leaves (altogether 13 nodes) are checked and evaluated in the entire computation.

y_j of their children Y_j are used, i.e., it is computed as $\max(|A| + |B| - 1, 0) = \max(y_{2i-1} + y_{2i} - 1, 0)$. At negation child(ren) Φ_i the values based on the values of the grandchildren are used, i.e., the difference between 1 and its children's values (let them be |A| and |B| here) in case of negated disjunction. This is due to the formula $1 - |\min(|A| + |B|, 1)| = \max(1 - (|A| + |B|), 1 - 1) = \max((1 - |A|) + (1 - |B|) - 1, 0) = \max(y_{2i-1} + y_{2i} - 1, 0)$ which has a similar structure as formula (3). In the case of negated implication nodes, by comparing equations (2) and (4), the right child plays a similar role as a child of

a disjunction node, while for the left child 1 – its value should be used (in contrast with the left child of a disjunction node). In this latter case, therefore 1 – (1 – its value) = its value is used. In this way, node Γ gets its value as max($y_1 + y_2 + y_3 + y_4 - 3$, 0). However, each y_i could have a value at most 1, which directly implies the cut condition given in the theorem. □

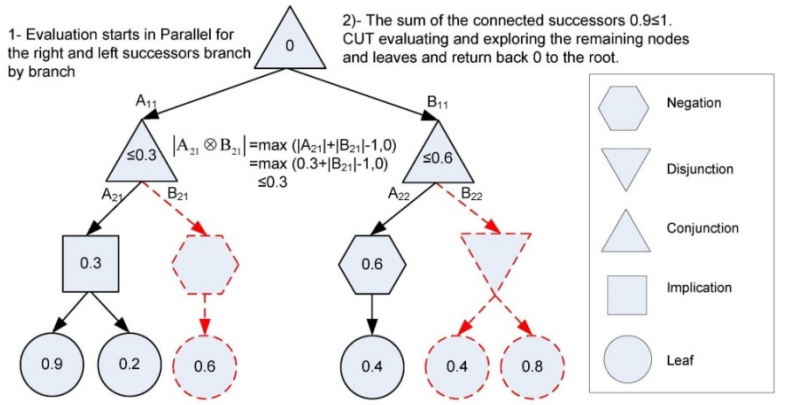

Figure 6

A pruning example applied at an L-and vertex having two L-and nodes children and their sum is not larger than 1. 5 operator nodes plus 3 leaves (altogether 8 nodes) out of 7 operator nodes plus 6 leaves (altogether 13 nodes) are explored and evaluated in the entire computation.

In special cases, the cut can be applied after evaluating two of the vertices Y_i having the sum of their value not more than 1. Figure 6 shows an example, where, actually, Y_1 and Y_3 are evaluated.

3.2 Implication Pruning

Consider that the root or the root of a subtree is an implication vertex. Various cuts can also be applied at these vertices. Let us start with the obvious ones.

Theorem 5. Let Π be an implication vertex. If its first (left) child has a value of 0 or its second (right) child is evaluated to have a value 1, then one can apply a lazy evaluation to assign a value of 1 to Π without checking and evaluating its other child.

Proof. From equation (2), the value of Π is at least 1 – | A |, where | A | is the value of its left child. This proves the first type of cut described by the theorem, substituting the value | A | = 0. From (2), it is also clear that the value of the implication Π cannot be less than the value of its right child | B |. Having | B | = 1 leads to the second cut technique described in the theorem. □

An example is depicted in Figure 7. It is shown that the evaluation of the left child resulted in a 0; it follows that the right child cannot have a lower value than this. Thus, the right child with all its subtrees can be pruned, and the value 1 can be returned to the root (implication) node without needing the value of the pruned subtree.

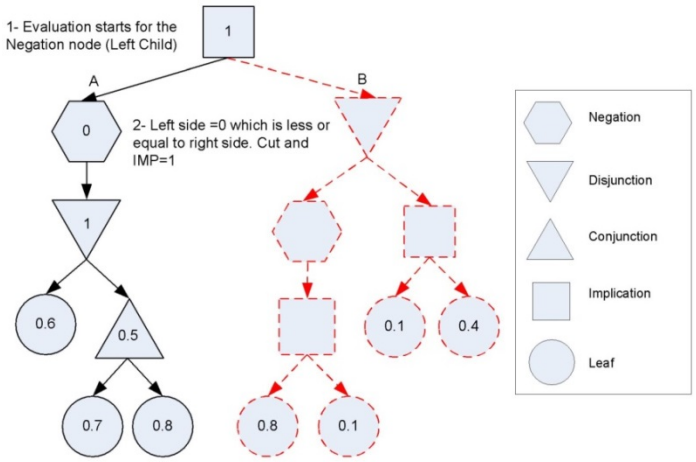

Figure 7

A pruning example for an implication node with the left child as a negation node. 4 operator nodes plus 3 leaves (altogether 7 nodes) out of 8 operator nodes plus 7 leaves (altogether 15 nodes) are explored and checked in the entire computation.

The next possible cut technique is more complex; the children of the implication vertex also play an important role.

Theorem 6. Let Π be an implication vertex such that its left child is either a

- negation vertex Φ_1 with a
  - L-or child Δ_1 or with an
  - implication child Π_1 or a
- L-and vertex Γ_1

and its right child is either a

- negation vertex Φ_2 with a L-and child Γ_2; an
- implication vertex Π_2 or a
- L-or vertex Δ_2.

Let Y_1 and Y_2 denote the children of the vertex Γ_1, Π_1 or Δ_1 depending on the cases above; and let Z_1 and Z_2 denote the children of the vertex Γ_2, Π_2 or Δ_2. Then, after

knowing the value of one of the Y_i and of the Z_j nodes, we may apply a cut as follows. For vertex Γ_1 let denote y_1 and y_2 the values of its children Y_1 and Y_2, respectively. For vertex Δ_1, let denote y_1 and y_2 the difference of 1 and its children's values. For Π_1, let y_1 denote the value of its left child Y_1 and let y_2 denote the difference of 1 and the value of its right child Y_2. Further, let z_1 and z_2 be the values of the children Z_1 and Z_2 of node Δ_2, respectively. In the case of node Γ_2, let denote z_1 and z_2 the difference of 1 and its children's, Z_1's and Z_2's, values. For Π_2, let z_2 denote the value of its right child Z_2 and let z_1 denote the difference between 1 and the value of its left child Z_1. If there is a value y_i which is not more than a value of a z_j ($i,j \in \{1,2\}$), then the value of Π is 1 and does not depend on the values of the other children's, i.e., on $y_{(3-i)}$ and $z_{(3-j)}$. Without evaluating these unnecessary parts, a cut can be applied.

Proof. Evaluating the implication expression (2), having the value of its children A and B, it has always an output which is not less than the value $1 - |A|$ and also than $|B|$, moreover it gets value 1 if and only if $|A| \leq |B|$. Let us consider the L-and node Γ_1 as a kind of MINIMUM node, and the L-or vertex Δ_2 as a kind of MAXIMUM node, while the opposite is true for them in the negated case, i.e., for nodes Γ_2 and Δ_1, respectively. Also in the negated case, instead of the values of the children, their 'opposite value', i.e., 1 – their values are used. Similarly, implication node Π_2 can be seen as a MAXIMUM node, but here for the left child 'opposite value' is used (and correspondingly Π_1 is like a MINIMUM node, and the 'opposite value' of the right child is used since it is a negated case). Thus, in each case listed in the theorem, the implication vertex Π has a left MINIMUM and a right MAXIMUM children. At a MINIMUM node X_1, if the value of one of the children is known, i.e., y_1 or y_2, the value of the X_1 cannot be larger than this value. For a MAXIMUM vertex X_2, its value cannot be less than any of the values z_1 and z_2 coming from its children. Let us see how we can combine this information at node Π. Then, the evaluation could start in parallel for one of the children of X_1 and one of the children of X_2, (e.g., for their left children, Y_1 and Z_1). While evaluating one of the successors of the left and right children, if the value at the right child has a larger or equal value to the value found at the left child, we can make a cut: we can assign value 1 to Π and we can stop evaluating and exploring the remaining parts of the subtree rooted at Π. □

We illustrate some of the cases of the possible cut described above by examples. Figure 8 depicts an implication node with a negated L-or vertex as its left child and another L-or as its right child. When we evaluate the children of both L-or nodes simultaneously after we saw the value of their left children, we got the information that the negated L-or (at the negation node) connected to the left is not larger than 0.4, while the L-or vertex connected to the right is not less than 0.7. So, it follows that the right child is already larger than the left one. To investigate and assess the right child of these L-or nodes is needless. The pruning is done and value 1 is returned at the root, i.e., at the implication vertex. For an example of implication

pruning in case the left child is an L-and and the right child is a negated L-and, see Figure 9.

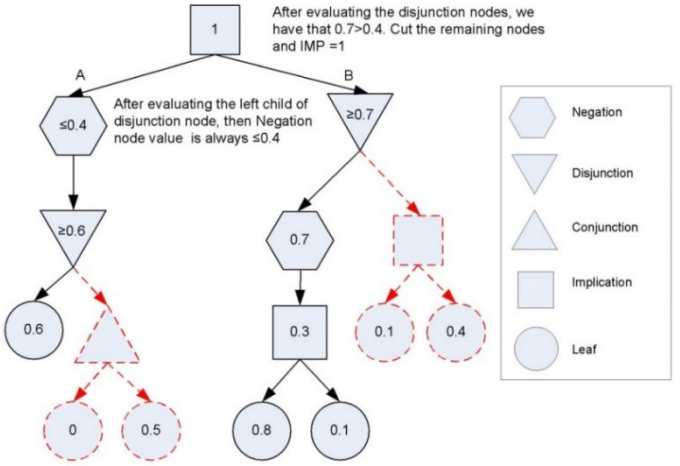

Figure 8

Example of an implication pruning with negated L-or (left child) and L-or (right child). 6 operator nodes plus 3 leaves (altogether 9 vertices) out of 8 operator nodes plus 7 leaves (altogether 15 vertices) are explored and evaluated in the entire computation.

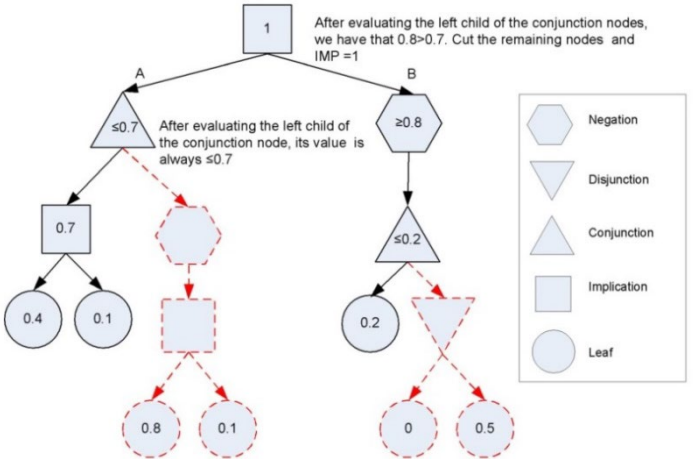

Figure 9

Example of an implication pruning with L-and and negated L-and. 5 operator nodes plus 3 leaves (altogether 8 vertices) out of 8 operator nodes plus 7 leaves (altogether 15 vertices) are explored and evaluated in the entire computation.

In our next example, it is shown how an implication node with an L-and vertex on the left and an L-or on the right is evaluated. The value of the L-or vertex becomes not less than the value of the L-and vertex, therefore we prune the rest, and return value 1 to the root (the implication node). Figure 10 shows a specific example.

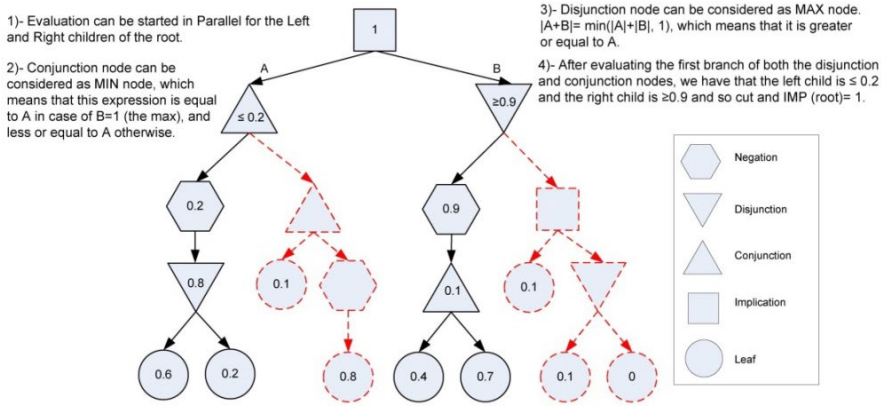

Figure 10

Evaluating implication with pruning (L-and is on the left and L-or is on the right). 7 operator nodes plus 4 leaves (altogether 11 nodes) out of 11 operator nodes plus 9 leaves (altogether 20 nodes) are explored and evaluated in the entire computation.

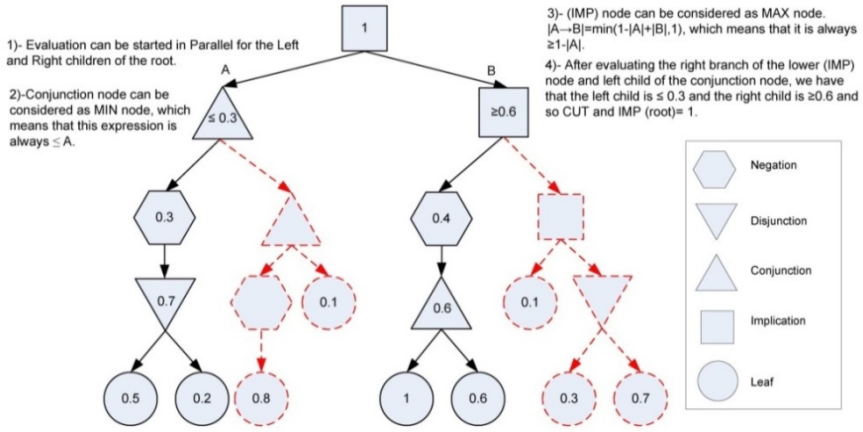

Figure 11

Evaluating implication vertex with pruning when one child is an L-or and the other child is an implication. 4 operator nodes plus 5 leaves (9 vertices in total) from the 20 vertices can be left out from the evaluation to compute the result

The last pruning example of this section has an implication child on the right side. When the first successors of the left and right children are evaluated, and we have found that the value at the right child has a larger or equal value to the value found at the left child, the pruning can be applied, thus, we can stop exploring and evaluating the rest of the nodes and the result 1 can be assigned to the root. Figure 11 presents the example.

4 Complex Examples

In this section, we see how the proposed pruning techniques can speed up the evaluation of complex examples in Łukasiewicz logic. The first example is shown in Figure 12, to evaluate this expression, without any pruning, twenty-eight vertices (leaves and operator nodes) are explored and evaluated to have the final result of the root.

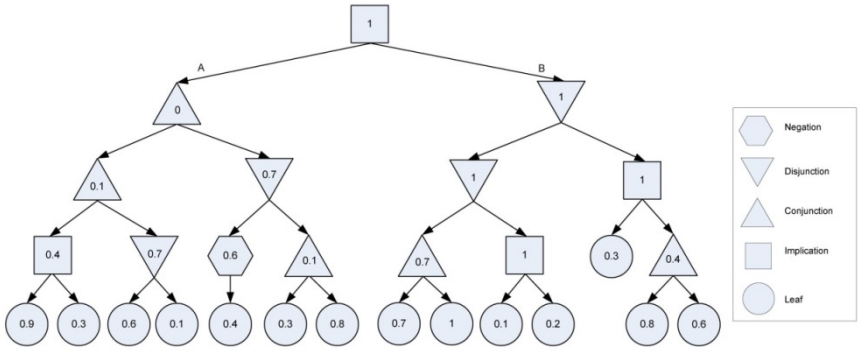

Figure 12

An example of a complex Łukasiewicz logic expression tree. 14 operator nodes plus 14 leaves (altogether 28 vertices) are explored and evaluated in the entire computation.

In contrast, after applying the proposed pruning techniques, only eleven vertices are explored and evaluated to get the same result at the root. See Figure 13 for the details about the cuts.

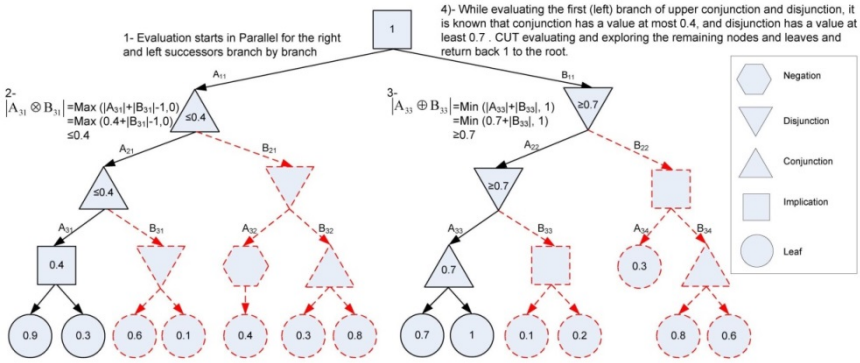

Figure 13

The expression tree of Figure 12 is evaluated by applying proposed pruning techniques. Only 7 operator nodes plus 4 leaves (altogether 11 nodes) out of 14 operator nodes plus 14 leaves (altogether 28 nodes) are explored and evaluated in the entire computation.

Figure 14 shows another example where forty-five nodes are explored and evaluated in the computation, without any pruning. Figure 15 shows that after applying our proposed cut methods, only twenty nodes are explored and evaluated to obtain the same result and assign it to the root.

As we have shown, the pruning techniques presented here can be applied very efficiently to reduce the cost of the evaluation. In a large formula, usually, a larger percentage of the formula can be cut.

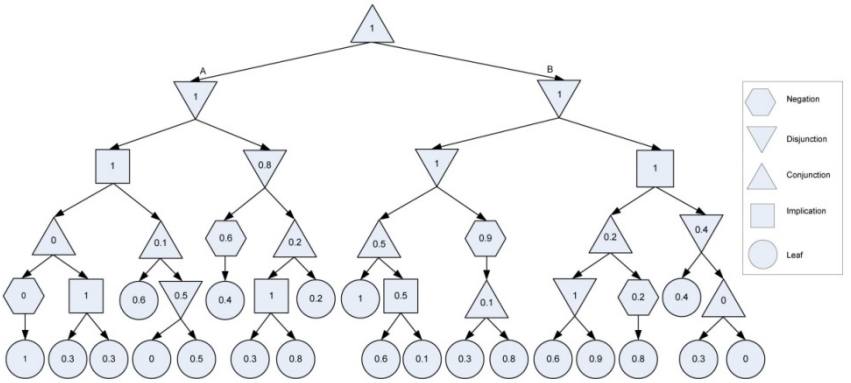

Figure 14
A complex Łukasiewicz logic expression tree. 24 operator nodes plus 21 leaves (altogether 45 vertices) are explored and evaluated in the entire computation.

Conclusions, Related and Future Work

Evaluation of logical formulas has many practical applications. If classical logic is considered, it is used in the hardware industry in logic gates, and it is used in the software industry and compiler techniques: programming languages apply short circuit evaluations. Fuzzy logics and fuzzy technology are very popular in various engineering solutions. Herein, we have proven pruning techniques for Łukasiewicz logic. Some other pruning techniques are also proposed for the other two fuzzy systems, for the Gödel and the product logics in our earlier papers [21] [22]. Although the general idea is similar for these techniques, different logical systems have different pruning methods.

As an important difference, let us compare the definitions of conjunction and disjunction of Gödel logic, the product logic, and the Łukasiewicz logic [21] [22]. Evaluating the same formula with only conjunctions and with the same values at each leaf, at Gödel logic and at product logic, one can apply a cut only if a leaf with value 0 is found (similarly to the Boolean case and also to the case of Theorem 1).

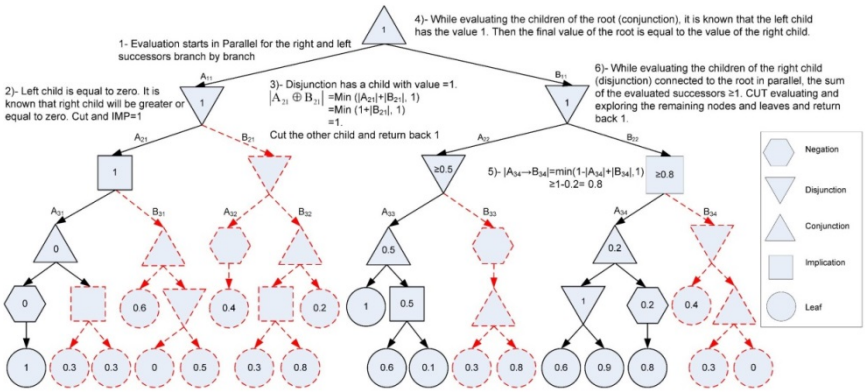

Figure 15

The tree of the formula of Figure 14 is evaluated by our proposed pruning techniques. Only 13 operator nodes plus 7 leaves (total 20 vertices) out of 24 operator nodes plus 21 leaves (altogether 45 vertices) are explored and evaluated in the entire computation.

Contrastingly, as we have seen it, e.g., in Theorem 4, in Łukasiewicz logic we can make a cut even if the above condition does not hold (e.g., by checking two leaves, let us say with values 0.3 and 0.65). When formulas having only disjunction operations are considered, at Gödel and product logics, to make a cut one needs to find a leaf with value 1 (similarly as we have stated in Theorem 2). However, in Łukasiewicz logic, (based on Theorem 3) cut can be applied without this condition, e.g., by checking leaves with values 0.4, 0.5, and 0.3. In [23], based on the theoretical results shown and proven here, a recursive algorithm has been presented that, similarly to the alpha-beta pruning for games, evaluates formula trees with arbitrary height. Also, a large number of simulations are presented there to prove the efficiency of the cut techniques; in practice, the method was tested on expression trees up to 100000 nodes.

Based on those facts, we believe that the presented pruning techniques yield very efficient tools for scientists, programmers, etc., working with fuzzy (Łukasiewicz) logic to make decisions faster and compute results with less effort. As future work, we may recall that fuzzy logic systems are generalized by using interval-values in [19] [24]. Efficient evaluation techniques in this interval-valued logic should also be investigated. There is also an interesting task to work with some kinds of generalizations of decisions and games, e.g., [17, 25, 26], as well as, extending the set of logical connectives in a programming language [27] and work on various evaluation techniques connected to other types of trees.

References

[1] Nagy, B.: Many-Valued Logics and the Logic of the C Programming Language. Proc. of ITI 2005: 27th International Conference on Information Technology Interfaces (IEEE), Cavtat, Croatia, 2005, pp. 657-662

[2] Bell, J., Machover, M.: A Course In Mathematical Logic. North-Holland, New York and Oxford, 1977

[3] Godo, L., Gottwald, S.: Fuzzy Sets and Formal Logics. Fuzzy Sets and Systems, Vol. 281, 2015, pp. 44-60

[4] Hájek, P.: Metamathematics of Fuzzy Logic. Trends in Logic, Vol. 4, Dordrecht: Kluwer Academic Publishers, 1998

[5] Barwise, J., Etchemendy, J.: The liar: An Essay on Truth and Circularity. New York: Oxford University Press, 1987

[6] Gödel, K.: Zum intuitioonischen Aussagenkalkül. Anzeigner Akademie der Wissenschaften im Wien, Mathematish-Naturwissenschaftliche Klasse, 69, 65-66, 1932; "On the Intuitionistic Propositional Calculus", reprinted in Kurt Gödel, Collected Works, Vol. 1, New York: Oxford Univ. Press, 1986

[7] Gottwald, S.: Many-Valued Logic. The Stanford Encyclopedia of Philosophy (Spring 2015 Edition), Edward N. Zalta (ed.), URL: http://plato.stanford.edu/entries/logic-manyvalued/, (First published Tue Apr 25, 2000; substantive revision Thu Mar 5, 2015)

[8] Łukasiewicz, J.: Selected Works Studies in Logic and The Foundations of Mathematics. North-Holland, Amsterdam, 1970

[9] Cignoli, R., D'Ottaviano, I., Mundici, D.: Algebraic Foundations of Many-valued Reasoning. Trends in Logic (Studia Logica Library), Vol. 7, Dordrecht: Springer, 2000

[10] Hähnle, R.: Many-valued logic and mixed integer programming. Annals of Mathematics and Artificial Intelligence, Vol. 12, 1994, pp. 231-264

[11] Zadeh, L.: Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi A. Zadeh. (G. J. Klir and B. Yuan, Eds.,) Advances in Fuzzy Systems – Applications and Theory, Vol. 6, World Scientific, River Edge, NJ, USA, 1996

[12] Zadeh, L.: Fuzzy logic—a Personal Perspective. Fuzzy Sets and Systems, Vol. 281, 2015, pp. 4-20

[13] Bagheri, S., Moniri, M.: Preservation theorems in Łukasiewicz model theory. Iranian Journal of Fuzzy Systems, Vol. 10, 2013, pp. 103-113

[14] Sayed, O., Borzooei, R.: Soft topology and soft proximity as fuzzy predicates by formulas of Łukasiewicz logic. Iranian Journal of Fuzzy Systems, Vol. 13, 2016, pp. 153-168

[15] Rich, E., Knight, K.: Artificial Intelligence. New York McGraw-Hill, 1991

[16] Russell, R., Norvig, P.: Artificial Intelligence, a Modern Approach. New Jersey: Prentice-Hall, 2003

[17] Basbous, R., Nagy, B.: Generalized Game Trees and their Evaluation. Proc. of CogInfoCom 2014: 5th IEEE International Conference on Cognitive Infocommunications, 2014, pp. 55-60. Vietri sul Mare, Italy

[18] Melkó, E., Nagy, B.: Optimal Strategy in Games with Chance Nodes. Acta Cybernetica, Vol. 18, 2007, pp. 171-192

[19] Nagy, B.: A General Fuzzy Logic Using Intervals. 6th Int. Symp. Hung. Researchers on Comp. Intell., Budapest, Hungary, 2005, pp. 613-624

[20] Kundu, S., Chen, J.: Fuzzy Logic or Lukasiewicz Logic: A Clarification. Fuzzy Sets and Systems, Vol. 95, 1998, pp. 369-379

[21] Basbous, R., Nagy, B., Tajti, T.: Short Circuit Evaluations in Gödel Type Logic. Proc. of FANCCO 2015: 5th Int. Conf. on Fuzzy and Neuro Computing, AISC Vol. 415, Hyderabad, India, 2015, pp.119-138, Springer

[22] Basbous, R., Tajti, T., Nagy, B.: Fast Evaluations in Product Logic. The 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2016), 2016, pp. 140-147, Vancouver, Canada

[23] Nagy, B., Basbous, R., Tajti, T.: Lazy evaluations in Łukasiewicz type fuzzy logic. *Fuzzy Sets and Systems*, Vol. 376, 2018, pp.127-151

[24] Nagy, B.: Reasoning by Intervals. Proc. of Diagrams 2006: Fourth International Conference on the Theory and Application of Diagrams, Stanford, CA, USA, LNCS-LNAI 4045, 2006, pp. 145-147

[25] Lakatos, G., Nagy, B.: Games with Few Players. Proc. of ICAI'2004: 6th Int. Conf. on Applied Informatics, Eger, Hungary, 2004, pp. II-187-196

[26] Basbous, R., Nagy, B.: Strategies to Fast Evaluation of Tree Networks. *Acta Polytechnica Hungarica*, Vol. 12, No. 6, 2015, pp. 127-148

[27] Nagy, B., Abuhmaidan, K., Aldwairi, M.: Logical conditions in programming languages: review, discussion and generalization. *Annales Mathematicae et Informaticae*, Vol. 57, 2023, pp. 65-77