New remarks regarding positive implicative BCK-algebras

Radu Vasile
Ovidius University
BLD Mamaia, 124, Constanţa
Romania
rvasile@gmail.com

Abstract. In this paper, we present a generalization of the Iseki Extension for a BCK-algebra. Furthermore, we prove that the Pseudo-Iseki extension is also a particular case of this generalization. Also, we introduce an algorithm for generating BCK-algebras using a computer and some new results regarding positive implicative BCK-algebras, results obtained with the help of this algorithm.

Keywords: BCK-algebras, positive-implicative BCK-algebras, BCK-trees, BCK-algebras extensions, generating algorithm.

1. Introduction

The concept of BCK-algebras appeared first in [6] as a generalization for the difference operation of two sets. Recently, in [7] the authors attached binary block codes to such an algebra. At the end of the article, they raised a question about the possibility to obtain a BCK-algebra starting from a binary block code. In [4], the authors have shown that in some special conditions, the answer to the question is yes, we can obtain BCK-algebras from binary block codes. Given these things, it has become interesting to study how BCK-algebras can be built and if there is a way to generate them using a computer. In [11], the author presents two new extensions for BCK-algebras and shows that the structure of the BCK-algebras class has the shape of a graph formed by three intersecting trees, named BCK-trees. The author also shows that BCK-algebras are divided in these trees by their properties. This means that we have the tree of commutative BCK-algebras, the tree of positive implicative BCK-algebras and that of implicative BCK-algebras. Moreover, the author shows that when two trees intersect each other, the BCK-algebra, which constitutes the intersection, loses properties, like in the Cayley Dickson derivation process for algebras over a field.

In ([8], pp. 249 - 256), the author constructs error-correcting codes attached to Hilbert algebras, based on the equivalence between positive implicative BCK-algebras and Hilbert algebras. In [5], starting from the equivalence between BCK bounded commutative algebras, Wajsberg and MV-algebras, the authors build binary block codes attached to Wajsberg and MV-algebras. In another paper, [1] authors give an algorithm for building Wajsberg Algebras, and, based on this,

in [2] the authors present an algorithm for building BCK bounded commutative algebras. In the first part of this paper, we present a new extension for BCK-algebras, a generalization for Iseki Extension and Pseudo-Iseki extension. In the second part, we show new identities that hold for positive implicative BCK-algebras. These identities were found using examples offered by a computer algorithm, which is presented in the last part of the paper.

2. Preliminaries

Definition 2.1 ([9]). *A set* $(X, *, 0)$ *that meets the following conditions:*

1. $((x * y) * (x * z)) * (z * y) = 0$, $\forall x, y, z \in X$;
2. $(x * (x * y)) * y = 0$, $x, y \in X$;
3. $x * x = 0$, $\forall x \in X$;
4. *If* $x * y = 0$ *and* $y * x = 0$, *then* $x = y$, $\forall x, y \in X$;

is a BCI-algebra

Definition 2.2 ([9]). *A BCI-algebra that meets the following relation is a BCK-algebra:*

5. $0 * x = 0$, $\forall x \in X$.

Proposition 2.1 ([10]). *Let* X *be a BCI-algebra. Then the following identities are true:*

6. $(x * y) * z = (x * z) * y$;
7. $(x * y) * x = 0$;
8. $((x * z) * (y * z)) * (x * y) = 0$;
9. $x * 0 = x$.

An alternative definition for the BCI-algebra and BCK-algebra is given in the following proposition:

Proposition 2.2 ([3], Proposition 3.1). *A BCI-algebra is defined also by the following axioms: 1, 4, 9,*
A BCK-algebra is defined by the following axioms: 1, 4, 9, 5.

Definition 2.3 ([9]). *A BCI-algebra* $(X, *, 0)$ *is commutative if the relation* $x * (x * y) = y * (y * x)$, *is true for all* $x, y \in X$.

Definition 2.4 ([10]). *A BCI-algebra* $(X, *, 0)$ *is positive-implicative, if the following relation is true:* $(x * z) * (y * z) = (x * y) * z$, *for all* $x, y, z \in X$.

Definition 2.5 ([9]). *A BCI-algebra $(X, *, 0)$ is implicative if the relation $x * (y * x) = x$ is true for all $x, y \in X$.*

Remark 2.1 ([9]). On a BCI-algebra, a partial order relation $<$ can be defined by $x < y$ if and only if $x * y = 0$.

Definition 2.6 ([9]). *Let X be a BCI / BCK-algebra. If there is an element 1 with $x < 1, \forall x \in X$, then X is called a bounded algebra.*

Definition 2.7 ([9]). *Let X be a BCK-algebra. It extends to a Bounded BCK-algebra if we add a 1 element that complies with the following rules:*

1. $1 * 1 = x * 1 = 0, \forall x \in X$;
2. $1 * x = 1, \forall x \in X$.

Remark 2.2 ([9]). If the BCK-algebra X is a positive implicative BCK-algebra, then the Iseki extension is a positive implicative BCK-algebra. If it is commutative, the Iseki extension is not commutative.

In [11], the author introduced a new Iseki like extension, named Pseudo-Iseki extension:

Definition 2.8 ([11]). *Let $(X, *, 0)$ be a BCK-algebra and I an element not in X. The $X[I]$ algebra, obtained using following rules is a BCK-algebra:*

1. $I * I = 0$;
2. $0 * I = 0$;
3. $x * I = x, \forall x \in X$;
4. $I * x = I, \forall x \in X$.

Remark 2.3 ([11]). If X is a commutative BCK-algebra, then the Pseudo-Iseki extension is commutative. If X is a positive implicative BCK-algebra, the Pseudo-Iseki extension is also positive implicative.

3. The generalized Iseki Construction of a BCK-algebra

Based on the new extension presented in [11], we will give in the following a generalized construction. We will show that the Iseki and Pseudo-Iseki extensions are particular cases of this generalized construction.

Proposition 3.1. *Let $(X, *, 0)$ be a BCK-algebra and x, y, z three elements in X with the following properties: $x * y = x$, $z * y = 0$. Then $x * z = x$.*

Proof. From axiom 1 of the BCK-algebra definition, we have: $((x * y) * (x * z)) * (z * y) = 0$. For the three elements mentioned, the expression is reduced to: $(x * (x * z)) = 0$, because $x * y = x$ and $z * y = 0$. In every BCK-algebra we have $(x * z) * x = 0$, whence $x = x * z$, according to the axiom 4 of BCK-algebras. □

Proposition 3.2. *Let X be a BCK-algebra and d an element not in X, with the following properties:*

1. $0 * d = 0$;
2. $d * d = 0$;
3. $d * x = d$, $\forall x \in X$;
4. $x * d = 0$ *or* $x * d = x$, $\forall x \in X$;
5. *If we have x in X with $x * d = 0$, then $(x * y) * d = 0$, $\forall y \in X$;*
6. *If there are two elements $x, z \in X$, with the following properties: $x * d = x$ and $z * d = 0$, then $x * z = x$.*

Algebra $X[D]$ is a BCK-algebra.

Proof. The axioms 3, 4, 5 are fulfilled from the construction process. It remains to prove the axioms 1 and 2. For the axiom 2, we have the following cases:

- $(x * (x * d)) * d$: $x * d$ can be x or 0. If $x * d = 0$, the entire expression becomes 0. If $x * d = x$, then the expression becomes $(x * x) * d = 0$;
- $(d * (d * x)) * x$: This is 0 because $d * x = d$ and $d * d = 0$.

For the axiom 1, we have the following expressions:

- $((d * y) * (d * z)) * (z * y)$: Is 0 because $d * y = d$, $d * z = d$ and $d * d = 0$;
- $((x * d) * (x * z)) * (z * d)$: We have multiple cases:
  If $x * d = 0$, the entire expression is 0. If $x * d = x$, the expression becomes: $(x * (x * z)) * (z * d)$. If $z * d = z$, then the expression is $(x * (x * z)) * z$, which is 0 from the axiom 2 of the algebra X. If $z * d = 0$ the expression is $x * (x * z)$. From the point 6 of the construction, if $x * d = x$ and $z * d = 0$, $x * z$ must be x, so the expression is $x * x = 0$;
- $((x * y) * (x * d)) * (d * y)$: $d * y = d$ and the expression is: $((x * y) * (x * d)) * d$. We have two cases: $x * d = x$ and $x * d = 0$. For $x * d = x$ the expression is: $((x * y) * x) * d$, which is 0, because in every BCK-algebra $(x * y) * x = 0$. If $x * d = 0$, the expression is $(x * y) * d$, which is 0 from the fifth point of the construction.
- The other cases are 0 because they contain identical elements on nearby positions. □

Definition 3.1. *The above construction will be named the generalized Iseki construction.*

Proposition 3.3. *If X is a positive implicative BCK-algebra, the BCK-algebra obtained through the generalized Iseki Construction, is a positive implicative BCK-algebra.*

Proof. We have the cases:

1. $(d*z)*(y*z) = (d*y)*z$: $d*y = d$, $d*z = d$ and the expression reduces to $d*(y*z) = d*z$. $d*z = d$, $d*(y*z) = d$, so the equality holds;

2. $(x*d)*(y*d) = (x*y)*d$: Here we have more cases: If $x*d = 0$, from the point 5 of the construction $(x*y)*d = 0$. If $x*d = x$ and $y*d = 0$, the expression reduces to $x = (x*y)*d$. But from point 6 of the construction, we have that: if $x*d = x$ and $y*d = 0$, then $x*y = x$. So the expression becomes $x = x*d$, having that $x*d = x$, the entire equality holds. If $x*d = x$ and $y*d = y$, the expression becomes $x*y = (x*y)*d$. But, we have already a BCK-algebra so the expression can be rewritten as $x*y = (x*d)*y$. But $x*d = x$, so the expression reduces to $x*y = x*y$;

3. $(x*z)*(d*z) = (x*d)*z$: $d*z = d$ and the expression becomes $(x*z)*d = (x*d)*z$. Which is equivalent with $(x*z)*d = (x*z)*d$;

4. $(d*d)*(y*d) = (d*y)*d$: $d*d = 0$, $d*y = d$, so the expression is 0 in both parts;

5. $(x*d)*(d*d) = (x*d)*d$: $d*d = 0$, and the expression becomes: $(x*d) = (x*d)*d$. We have two cases, $x*d = 0$, both parts becoming 0 and $x*d = x$, when the expression becomes $x = x*d$.

□

Proposition 3.4. *Let X be a commutative BCK-algebra. The BCK-algebra obtained through the generalized Iseki construction is not always commutative.*

Proof. We have the following expressions:

1. $x*(x*d) = d*(d*x)$: $d*x = d$, $d*d = 0$, so in the right hand we have 0. $x*d$ can be x, whence $x*x = 0$, and the expression holds. $x*d$ can be 0, but $x*0 = x$, and the expression does not hold anymore;

2. $d*(d*y) = y*(y*d)$: In this case $d*y = d$ and $d*d = 0$ and $y*d$ can be$y$, in which case $y*y = 0$ and the expression holds. But, if $y*d = 0$, $y*0 = y$ and the expression does not hold any more. □

Remark 3.1. Let X be a commutative BCK-algebra and Y the BCK-algebra obtained from X through the generalized Iseki Construction. Y is commutative if and only if for all $x \in X$, $x*d = x$. Reformulating, y is commutative if there is not any x for which $x*d = 0$.

Remark 3.2. Let X be a commutative BCK-algebra and Y the BCK-algebra obtained from X through the generalized Iseki Construction. If $x * d = 0$, for all $x \in X$, we have the Iseki Extension of X. If $x * d = x$ for all $x \in X$, we have the Pseudo-Iseki extension of X (see [11]).

Remark 3.3. From the above remarks, we observe that the Pseudo-Iseki extension is the only case that maintains commutativity. On the other hand, we have noticed that the generalized Iseki Constructions can not be applied to any BCK-algebra, in total contrast with its particular cases.

Indeed, to the following BCK-algebra, we can not apply one of the cases of the generalized Iseki construction:

Alg 1.

$*$	0	1	2
0	0	0	0
1	1	0	0
2	2	1	0

Alg 2.

$*$	0	1	2	3
0	0	0	0	0
1	1	0	0	0
2	2	1	0	2
3	3	3	3	0

, because this is not longer a BCK-algebra.

Remark 3.4. The generalized Iseki construction can be applied to all BCK-algebras from the 2nd and 3rd BCK-trees (see, [11]), the BCK-trees that contains positive implicative BCK-algebras while they do not intersect with the 1st BCK-tree. Thus, this construction can be applied while the source BCK-algebra does not contain a sub-algebra of order 3 isomorphic to the BCK-algebra from Alg 1.

4. New results about positive implicative BCK-algebras

In the following, we will present some new relations about elements in positive implicative BCK-algebras and an interesting way for building BCK-algebras. In [4], Proposition 2.1, a set with a binary relation is taken. There the author shows that, in some conditions, the set is a non-commutative, non-implicative BCK-algebra.

Proposition 4.1 ([4])**.** *Let A be a partial ordered set, and $*$ a binary relation on A with the following properties:*

1. $0 * x = 0$;
2. $x * x = 0$;
3. $x * y = 0$ *if and only if* $x \leq y$;

4. $x * y = x$ *otherwise;*

A together with $*$ *is a positive implicative BCK-algebra.*

Proof. First, we should prove that this set is a BCK-algebra, then we should prove that is a positive implicative one. From the conditions, we observe that the third, fourth and fifth axioms are fulfilled. The second axiom says that $(x * (x * y)) * y = 0$. Here we have two cases:

1. $x * y = 0$: $(x * (x * y)) * y = (x * 0) * y = x * y = 0$;
2. $x * y = x$: $(x * (x * y)) * y = (x * x) * y = 0 * y = 0$.

Now we arrive to the first axiom: $((x * y) * (x * z)) * (z * y) = 0$. There are two cases:

1. $x * y = 0$: $((x * y) * (x * z)) * (z * y) = (0 * (x * z)) * (z * y) = 0$;
2. $x * y = x$: $((x * y) * (x * z)) * (z * y) = (x * (x * z)) * (z * y)$. Here we also have two cases:
   (a) $x * z = 0$: $(x * (x * z)) * (z * y) = x * (z * y)$. In this moment we have that y is not greater than x, but z is greater than x. So y can not be greater than z, because then y would be greater than x. Whence $z * y = z$. $x * (z * y) = x * z = 0$;
   (b) $x * z = x$: $(x * (x * z)) * (z * y) = (x * x) * (z * y) = 0$.

Now we should prove that this is a positive implicative BCK-algebra. For this we should have $(x * z) * (y * z) = (x * y) * z$. We have two cases:

1. $x * z = 0$: $(x * z) * (y * z) = 0 * (y * z) = 0$. $(x * y) * z = (x * z) * y = 0 * y = 0$;
2. $x * z = x$: $(x * z) * (y * z) = x * (y * z)$. $(x * y) * z = (x * z) * y = x * y$. In this case, z is not greater than x. On the other hand, we can have $y * z = 0$ and $y * z = y$. In the last case, we obtain $x * y = x * y$. In the case when $y * z = 0$, we have $x = x * y$. But in this case $x * y$ can not be 0, because that means $x \leq y \leq z$ which contradicts our hypothesis $x * z = x$. So, $x * y = x$ in this case and the proof is complete. □

The interesting question is, when the converse is true, in what conditions a positive implicative BCK-algebra has this explicit general form. At a first sight, the answer is simple, using that in any BCK-algebra $x * y \leq x$, to obtain the desired form we should impose that $x * (x * y) = 0$ and $y * (y * x) = 0$, or $x * (x * y) = x$ and $y * (y * x) = 0$. For the last part the symmetric case is also possible: $x * (x * y) = 0$ and $y * (y * x) = y$. However, a deeper look shows us some interesting things about positive implicative BCK-algebras and the behaviour of their elements. In particular, these new things give us the possibility to build some higher-order positive implicative BCK-algebras. For this, we will give in the following a theorem with new relations regarding positive implicative BCK-algebras.

Theorem 4.1. *Let X be a positive implicative BCK-algebra. The following relations are true for any $x, y \in X$:*

1. $x * (x * y) \leq y * (x * y)$;
2. $x * (y * (x * y)) = x * y$;
3. $(y * (y * x)) * (y * (x * y)) = y * (y * (x * y))$;
4. $(x * (x * (y * x))) * (x * y) = (x * (x * (y * x)))$;
5. $(x * (x * y)) * (y * (y * x)) = x * (x * (y * x))$;
6. $x * (y * (y * (x * y))) = x$;
7. $X * (x * (y * x)) = x * (x * (y * (x * (x * y))))$;
8. $(x * (y * (x * (x * y)))) = x * (y * x)$.

Proof. 1. From the positive implicative BCK-algebra definition we have: $(x * (x * y)) * (y * (x * y)) = (x * y) * (x * y) = 0$. The symmetric relation is also true: $(y * (y * x)) \leq x * (y * x)$;

2. $y*(x*y) \leq y$. Hence $x*y \leq x*(y*(x*y))$. But $x*(y*(x*y))*(x*y) = (x*(x*y))*(y*(x*y))$. From the point 1, this is 0 and, by means of the fourth axiom, it results that $x * (y * (x * y)) = (x * y)$.

3. $(y*(y*(x*y)))*((y*(y*x))*(y*(x*y)))= (y*(y*(y*x)))*(y*(x*y))= (y * x) * (y * (x * y)) = 0$. The other inequality: $((y * (y * x)) * (y * (x * y))) * (y * (y * (x * y)))= (y * (y * x) * y) * (y * (x * y)) = 0$.

4. $(x * (x * y)) * (x * (y * x))= (x * (x * (y * x))) * (x * y)\leq x * (x * (y * x))$. Conversely:

   $(x*(x*(y*x)))*((x*(x*y))*(x*(y*x)))= (x*(x*(x*y)))*(x*(y*x))\leq (x * y) * (x * (y * x)) \leq 0$.

5. $(x * (x * y)) * (y * (y * x))$ can be written as: $(x * (y * (y * x))) * (x * y)$. $(x * (x * (y * x)))$ can be written as: $(x * (x * (y * x)) * (x * y)$. $(x * (y * (y * x)))*(x*y)*(x*(x*(y*x)))= ((x*(y*(y*x)))*(x*(x*(y*x))))*(x*y)= ((x*(x*(x*(y*x))))*(y*(y*x)))*(x*y)= ((x*(y*x))*(y*(y*x)))*(x*y)= ((x * y) * (y * x)) * (x * y)= ((x * y) * (x * y)) * (y * x) = 0$. Conversely: $((x*(x*(y*x)))*(x*y))*((x*(y*(y*x)))*(x*y)) = ((x*(x*(y*x)))*(x*(y*(y*x))))*(x*y) \leq ((y*(y*x))*(x*(y*x)))*(x*y) \leq 0*(x*y) = 0$.

6. According to the first BCK-algebra axiom, we have: $(y*(y*(x*y))) \leq y*(y*x)$. From [9] we have that: $x*(y*(y*x)) \leq x*(y*(y*(x*y))) \leq x$. From the same result we have that: $x*(x*(y*(y*(x*y)))) \leq x*(x*(y*(y*x)))$. On the other hand: $x*(x*(y*(y*x))) \leq x*(y*(y*(x*y)))$. So, we have

that: $(x*(x*(y*(y*(x*y)))))*(x*(x*(y*(y*x)))) = 0$. At the same time, $(x*(x*(y*(y*(x*y)))))*(x*(x*(y*(y*x)))) = (x*(x*(y*(y*(x*y))))))$. Therefore: $(x*(x*(y*(y*(x*y)))))) = 0$, whence $x = (x*(y*(y*(x*y))))$.

7. $(x*(x*(y*x)))*(x*(x*(y*(x*(x*y))))) \leq (x*(y*(x*(x*y))))*(x*(y*x)) \leq (y*x)*(y*(x*(x*y))) \leq (x*(x*y))*x = 0$. Conversely: $(x*(x*(y*(x*(x*y))))) = (x*(x*(y*(x*(x*y)))))*(x*y)$. $(x*(x*(y*x))) = (x*(x*(y*x)))*(x*y)$. $((x*(x*(y*(x*(x*y))))))*(x*y))*((x*(x*(y*x))))*(x*y) = ((x*(x*(y*(x*(x*y))))))*(x*(x*(y*x))))*(x*y)$. $((x*(x*(y*(x*(x*y))))))*(x*(x*(y*x))))*(x*y) \leq ((x*(y*x))*(x*(y*(x*(x*y)))))*(x*y) \leq ((y*(x*(x*y)))*(y*x))*(x*y) \leq (x*(x*(x*y)))*(x*y) \leq (x*y)*(x*y) = 0$.

8. It emerges from Point 7 if we multiply with x in both parts. □

Remark 4.1. As we have proven the above relations for any x, y, by changing x with y, we get the symmetric relations being true too.

Proposition 4.2. *Let X be a positive implicative BCK-algebra and $x, y \in X$ with the property that $x * y = 0$, then $y * (y * x) = x * (y * x)$.*

Proof. From Theorem 4.1 we have: $(y*(y*x)) \leq (x*(y*x))$. From the first axiom of BCK-algebras, we have already that: $((x*(y*x))*(y*(y*x))) \leq (x*y)$, thus $((x*(y*x))*(y*(y*x))) \leq 0$. From the fifth axiom, we have that the above expression is 0. From the fourth axiom, it results that the two expressions are equal. □

Remark 4.2. The equality can be obtained directly from the positive implicative BCK-algebra definition as follows: $((x*(y*x))*(y*(y*x))) = (x*y)*(y*x)$. Through the chosen proof, we have shown that the second inequality is true in any BCK-algebra, for any two elements x, y with the property that $x*y = 0$. this inequality becomes equality only when the BCK-algebra is a positive implicative one, for the elements x, y with the property that $x * y = 0$.

Remark 4.3. The result can also be obtained from Theorem 4.2, Point F from [9], where we have that in a positive implicative BCK-algebra $(y*(y*x))*(x*y) = (x*(x*y))*(y*x)$, by taking $(x*y) = 0$.

Now, We will see how the elements of a positive-implicative BCK-algebra behave in certain conditions.

Proposition 4.3. *Let X be a positive implicative BCK-algebra and $x, y \in X$ with $x*y = 0$, $y*x \neq 0$ and $x*(y*x) \neq 0$. The subalgebra generated by x, y is of order 6 and contains a subalgebra of order 4 which is bounded implicative.*

Proof. We have that $x*y = 0$, $y*x \neq 0$ and $x*(y*x) \neq 0$. We will use the following notations:

1. $1 = x * (y * x) = y * (y * x)$ from Proposition 4.2;
2. $2 = x * (x * (y * x))$;
3. $3 = x$;
4. $4 = y * x$;
5. $5 = y$;

Using these elements, together with 0 and the relations from theorem 4.1, we obtain a BCK-algebra of order 6 with the following multiplication table:

Alg 3.

$*$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	1	0	1	0	1	0
2	2	2	0	0	0	0
3	3	2	1	0	1	0
4	4	4	4	4	0	0
5	5	4	5	4	1	0

This algebra is a BCK-algebra of order 6. Moreover, we see that it contains a bounded implicative BCK-algebra of order 4. □

Remark 4.4. How we notice, the algebra from Alg 3, is not in the desired form, the one specified in 4.1. The condition used in the statement of the previous proposition made this happen. If we take $x, y \in X$, with $x * y = 0$ and $x * (y * x) = 0$, this implies that $y * (y * x) = 0$ and the multiplication table modifies as follows:

Alg 4.

$*$	0	3	5
0	0	0	0
3	3	0	0
5	5	5	0

This is a BCK-algebra of order 3, which has the desired form. Reformulating, when $x * y = 0$ $y * x$ must be y. Any other combination of elements does not offer us the desired form. For example if we take $x, y \in X$ with $x * y = 0$ and $x * (y * x) = x$ the table will look like:

Alg 5.

$*$	0	3	4	5
0	0	0	0	0
3	3	0	3	0
4	4	4	0	0
5	5	4	3	0

This is a bounded implicative BCK-algebra of order 4.

We have seen until now what happens in a positive implicative BCK-algebra when we have two elements x, y with the property that $x * y = 0$. Now, we should see which is the behaviour of two elements that have no relation between them.

Proposition 4.4. *Let X be a positive implicative BCK-algebra and $x, y \in X$ two elements with the property $x * y \neq 0$ and $y * x \neq 0$. If $x * (y * x) \neq (x * y)$ and $x * (y * x) \neq x$, then the subalgebra generated by x, y is a BCK-algebra of order 14 which contains a bounded implicative subalgebra of order 4.*

Proof. To prove this, we will use the relations from Theorem 4.1 and other known results about positive - implicative BCK-algebras, mentioned in [9]. We will use the following notations:

1. $1 = y * (y * (x * y))$;
2. $2 = (y * (y * x)) * (x * y)$;
3. $3 = y * (y * x)$;
4. $4 = (x * y)$;
5. $5 = x * (y * x)$;
6. $6 = x * (x * (y * x))$;
7. $7 = x * (x * y))$;
8. $8 = x * (y * (y * x))$;
9. $9 = x$;
10. $10 = y * x$;
11. $11 = y * (x * y)$;
12. $12 = y * (x * (x * y))$;
13. $13 = y$;

Together with the element 0, after some computations, we obtain the following multiplication table:

Alg 6.

$*$	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	1	0	0	1	1	0	0
2	2	2	0	0	2	0	2	0	2	0	2	0	2	0
3	3	2	1	0	2	0	3	1	2	0	3	1	2	0
4	4	4	4	4	0	0	4	4	0	0	4	4	4	4
5	5	5	4	4	2	0	5	4	2	0	5	4	5	4
6	6	6	6	6	6	6	0	0	0	0	0	0	0	0
7	7	7	6	6	7	6	2	0	2	0	2	0	2	0
8	8	8	88				4	4	0	0	4	4	4	4
9	9	9	8	8	7	6	5	4	2	0	5	4	5	4
10	10	10	10	10	10	10	10	10	10	10	0	0	0	0
11	11	11	10	10	11	10	11	10	11	10	2	0	2	0
12	12	10	12	10	10	10	12	12	10	10	1	1	0	0
13	13	11	12	10	11	10	13	12	11	10	3	1	2	0

This is a BCK-algebra of order 14, which contains a bounded implicative subalgebra of order 4. □

Remark 4.5. We notice that the algebra in Alg 6 is not in the desired form, the one specified in 4.1. The form specified there says that $x*y$ should be x if it is not 0. So, in the above table, we will take $x * y = x$ and see how the table modifies. If $x * y = x$ this means that $x * (x * y) = 0$ which implies that any element less than 7, is 0. This means that 2 and 6 are 0. Also, $x*(y*x) = x*(y*(y*x)) = x$ so $4 = 5 = 8 = 9$. The implied relation makes $1 = 3$, $11 = 10$ and $12 = 13$. Therefore $1, 2, 4, 5, 6, 7, 8, 11, 12$ will disappear from the table.

It will remain a BCK-algebra of order 5:

Alg 7.

$*$	0	3	9	10	13
0	0	0	0	0	0
3	3	0	0	3	0
9	9	9	0	9	9
10	10	10	10	0	0
13	13	10	10	3	0

How we see, the algebra is not in the desired form. For this, we should go further and impose that $y * (y * x) = 0$. This makes 3 and 10 disappear from the table.

Alg 8.

$*$	0	9	13
0	0	0	0
9	9	0	9
13	13	13	0

This way, we obtained an implicative BCK-algebra of order 3, which has the desired form.

5. An algorithm to generate BCK-algebras of order n with $n \leq 6$ using a computer

In building the results from the previous sections, we used a computer algorithm, which, up to our knowledge, is the first algorithm for generating BCK-algebras. We will present in this section the algorithm we used and some improvements for it. This algorithm is based on findings presented in [11], where the author shows that the higher-order BCK-algebras can be built through extensions from the lower-order ones. Also, the author shows that there are three BCK-algebras of order 3, so every higher-order BCK-algebra must contain one or more of these BCK-algebras of order 3. In conclusion, the author shows that the BCK-algebras are organized on three trees, named BCK-trees, each tree having as a root one of the tree BCK-algebras of order 3. An important thing about these trees is that every tree contains BCK-algebras with distinct properties. Thus, we have the tree of commutative BCK-algebras (tree NO. 1), the tree of positive implicative BCK-algebras (tree NO. 2) and the tree of implicative BCK-algebras (tree NO. 3). The author shows that these three trees intersect each other. The intersection points are represented by Isomorphic BCK-algebras. Moreover, from each point of intersection, the higher-order BCK-algebras lose one of their properties. Starting from this point, we concluded that we could write an algorithm that starts with one of the three algebras of order 3 and builds the BCK-algebras of order 4. Once it has found all BCK-algebras of order 4, it takes each BCK-algebra of order 4 and builds the BCK-algebras of order 5, which contains that BCK-algebra of order 4. The technique resembles a Deep-first search for graphs. In [11], the author discusses the complexity of such an algorithm, showing that it is $O((N^{n-2})^2)$ for each level of BCK-algebras, the total complexity being the product of complexities for each level.

5.1 Algorithm description

For finding the BCK-algebras of order N, starting from a BCK-algebra of order $N-1$, the algorithm takes the multiplication table of that BCK-algebra, which is a table with $N-1$ lines and $N-1$ columns, and borders it with one new line and one new column. On this new line and new column, the corner elements have fixed values, according to the BCK axioms: $[N-1,0] = N-1$, $[0,N-1] = 0$, $[N-1,N-1] = 0$. So, on the new line and new column there remain $(n-2)$ positions for each, positions that can vary between 0 and $N-1$ (see, [11]). Also, in establishing these corner values, we used the alternative definition for BCI and BCK-algebras, see 2.2. This way, we build a candidate table, with N lines and N columns, that satisfies the axioms $2, 3, 5$ and $x * 0 = x$ (see 9). In the next step, the algorithm iterates between 0 and $N-1$ for each of these $(N-2)*(N-2)$ positions, building this way all possible tables that can contain the multiplication table of the specified BCK-algebra. In this iteration process, the algorithm verifies that the built table satisfies the fourth axiom by checking if there are zeros on correspondent positions on the new line and column. This

way, even from the construction process of the table, the constructed tables satisfy the axioms 2, 3, 4, 5 of the BCK-algebra definition. The only axiom that must be checked is 1, but this axiom should be checked only for the newly introduced element in relation with the other elements from the BCK-algebra of order $N-1$. It is not necessary to check again that the $N-1$ elements satisfy the 1st axiom among them because we know that already from the previous steps.

The code is written in the Python language. It consists of two classes, one for representing logical algebras, one for representing BCK-trees and the main script. The class for logical algebras serves as a base class for all other logical algebras, useful if we will want to extend this software to work with other types of logical algebras too. At this moment, this class builds a logical algebra and can check if the algebra is a BCI or BCK-algebra. Also, the class can check if the BCI or BCK-algebra is commutative, positive implicative, or implicative.

The BCK tree class receives an algebra as a start point and builds all BCK-algebras of greater order starting from this algebra it received. It generates algebras up to an order received as a parameter for the generating function. For the generation process, two recursive functions are used, one which generates each order of BCK-algebras and one which generates all possible tables for a specific order.

The main script builds the 3 BCK-algebras of order 3, builds 3 instances of BCK Tree class, one for each 3 order algebra, and calls for each one the generating function with a parameter representing the desired order. The generating function calls itself recursively for each level up to specified order.

5.2 Algorithm performance

This is the first version of the algorithm. Various improvements are possible. The first improvement we made is that we used multiprocessing to parallelize the process of BCK-tree generation. These tree BCK-trees can be generated in parallel, independent one from another. The algorithm could be parallelized further at the table generation process level. Another improvement could be the introduction of a storage component to store all the generated BCK-algebras. This way, when someone asks for BCK-algebras of a specific order, the algorithm will check if it already generated them. If yes, it will serve them to the user. If not, it will check which is the last order it generated and will start to generate the BCK-algebras from the next order up to the order demanded by the user. For example, if the user asks for BCK-algebras of order 9 and the algorithm finds that it has generated all the BCK-algebras up to order 7, it will start to generate the BCK-algebras of order 8 and 9.

We have run the algorithm on four different types of machines and configurations to generate BCK-algebras of order 4, 5, 6, both in sequential and multiprocessing executions. There were two runs, one asking for BCK-algebras

up to order 5, another asking for BCK-algebras up to order 6. The machines used are:

1. Raspberry pi 4 - Arm 72 CPU, four cores at 1.5 GHz, 4 GB of RAM, with SD card Class 10 US3 Storage, having Raspbian 10 as OS;
2. Laptop - Intel I5 3210M CPU, two cores at 2.5 GHz, 12 GB of RAM, with SSD storage, having Windows 10 as OS;
3. PC - Intel I5 2400, four cores at 3.1 GHz, 8 GB of RAM, with SSD storage, having Windows 10 as OS;
4. VPS - Intel Xeon E5 2400 CPU, six cores at 2.2 GHz, 16 GB of ram, with SSD storage, having UBUNTU 20.04 as OS;

The runing times, in seconds, are presented in the following table, where (P) means parallel and (S) means sequential:

Machine	5 order (P)	5 order (S)	6 order (P)	6 order (S)
Raspberry Pi	2.71	6.90	5131.58	12442.49
Laptop	3.55	5.55	3884.66	7033.84
PC	2.49	4.42	3722.74	7407.54
VPS	1.16	2.94	2057.84	4676.01

5.3 Algorithm source code

In the following, we present the algorithm source code.

Listing 1: Logical Algebra class

```
class LAlgebra(object):
    '''
    classdocs
    '''

    def __init__(self,dim,den,alg=None):

#the dimension of the algebra
        self.n=dim
#the name of the algebra (id)
        self.den=den
        #initialises the table of the algebra with 0
        self.t= [x[:] for x in [[0]\ast dim]\ast dim]
#copy the source algebra if one is given
        if alg is not None:
            for i in range(0,alg.n):
                for j in range(0,alg.n):
                    self.t[i][j]=alg.t[i][j]
```

```
            for i in range(alg.n,dim):
                for j in range(0,dim):
                    self.t[i][j]=0;
                    self.t[j][i]=0;

#displays the algebra as simple text
    def __str__(self):
            return
            ('\n'+self.den+'\n')+('\ast|'+(''.join(
            ['{:4}'.format(i)\
            for i in range(0,self.n)])))
            +'\n'+('-|'+(''.join(
            ['----' for i in range(0,self.n)])))
            +'\n'+('\n'.join(
            [str(row[0])+'|'+(''.join(
            ['{:4}'.format(item) for item in row]))\
             for row in self.t]))

            #checks  if the logical algebra
#satisfies the 5th axiom
#and x\ast0=x in the same time
    def CheckAxiomBCI59(self):
        for i in range(0,self.n):
            if self.t[0][i]!=0 or  self.t[i][0]!=i:
                return False
        return True

            #checks if the logical algebra satisfies
           #x\ast0=x
    def CheckAxiomBCI9(self):
        for i in range(0,self.n):
            if self.t[i][0]!=i:
                return False
        return True

#checks if a logical algebra satisfies the 4th BCK axiom
    def CheckAxiomBCI4(self):
        for i in range(0,self.n):
            for j in range(0,self.n):
                if i!=j and
                self.t[i][j]==0 and self.t[j][i]==0:
                    return False
        return True
```

```
#checks if a logical algebra
#satisfies the first BCK axiom
    def CheckAxiomBCI1(self):
        for i in range(0,self.n):
            for j in range(0,self.n):
                for k in range(0,self.n):
                    if self.t[self.t[
                       self.t[i][j]][
                       self.t[i][k]]][self.t[k][j]]!=0:
                           return False
        return True

#checks if an extension table
#of a BCK-algebra satisfies the first BCK axiom
    def CheckAxiomBCI1ForExt(self):
        for i in range(0,self.n):
            for j in range(0,self.n):
                #if i!=j:
                if i!=j and (
                self.t[self.t[
                self.t[self.n-1][i]][
                self.t[self.n-1][j]]][
self.t[j][i]] or
                self.t[self.t[
                self.t[i][self.n-1]][
               self.t[i][j]]][self.t[j][self.n-1]] or
                self.t[self.t[
                self.t[i][j]][self.t[i][self.n-1]]][
                self.t[self.n-1][j]]):
                    return False
        return True

#checks if the logical algebra is a BCK-algebra
    def IsBCK(self):
        return self.CheckAxiomBCI59() and
        self.CheckAxiomBCI4() and self.CheckAxiomBCI1()

#check if the Logical Algebra
# is a BCI-algebra
    def IsBCI(self):
        return self.CheckAxiomBCI9() and
        self.CheckAxiomBCI4() and self.CheckAxiomBCI1()
```

```
#checks if the BCK-algebra is commutative
    def isBCKCom(self):
        for i in range(0, self.n):
            for j in range(0, self.n):
                if(self.t[i][self.t[i][j]]
               !=self.t[j][self.t[j][i]]):
                    return False
        return True

#checks if the BCK-algebra is implicative
    def IsBCKImp(self):
        for i in range(0, self.n):
            for j in range(0, self.n):
                if(self.t[i][self.t[j][i]]!=i):
                    return False
        return True

#checks if the BCK-algebra is positive implicative
    def IsBCKPoz(self):
        for i in range(0,self.n):
            for j in range(0,self.n):
                for k in range(0,self.n):
                    if self.t[self.t[i][k]][self.t[j][k]]
                    !=self.t[self.t[i][j]][k]:
                        return False
        return True

#Checks the properties of the BCK-algebra
# and set them accordingly
    def CheckBCKProperties(self):
        self._bckcom=False
        self._bckimp=False
        self._bckpoz=False
        if self.IsBCKImp():
            self._bckcom=True
            self._bckpoz=True
            self._bckimp=True
            self.den=self.den+"_[I]"
        elif self.isBCKCom():
            self._bckcom=True
            self.den=self.den+"_[C]"
        elif self.IsBCKPoz():
            self._bckpoz=True
            self.den=self.den+"_[P}"
```

Listing 2: BCK Tree class

```
class BCKTree(object):
    def __init__(self,alg):
#the starting algebra for which the BCK-tree is build
        self.alg=alg
#the list of next level (nl) BCK-algebras
        self.nl=[]
#recursive function used to
#generate a table with N lines and N columns
#starting from a BCK-algebra of order N-1
#The function iterates from 0 to N-1
#for each position from the last line and column,
#checking if it has found a BCK-algebra
#s is the current free position
    def GenerateTable(self,s,a):
#the dimension of the source algebra
        d=a.n
#iterates over each value from 0 to n-1
        for i in range(0,d):
#if the function is on the last column free positions
            if s>d-2:
#checks if the 4th axiom is satisfied, namely
#the corespondent positions
#on the last line and the last column
#should not be 0 in the same time
#if there is already 0 on the last line, skip
                if i==0 and a.t[d-1][s+2-d]==0:
                    continue;
                #set the position: s+2-d:d-1
#on the last column
                a.t[s+2-d][d-1]=i

            else:
#set the position: d-1:s
#on the last line
                a.t[d-1][s]=i
#check if all the free positions received a value
            if s==2\ast d-4:
#check if the axiom 1 is satisfied by the table build
                if a.CheckAxiomBCI1ForExt():
                    pos=len(self.nl)-1
                    e=len(self.nl[pos])+1
#put the new found BCK-algebra in the list of algebras
                    self.nl[pos].append(
```

```
                    LAlgebra(d,a.den+'-'+str(e),a))
            else:
#if not all the free positions received a value,
# the function call itself for the next free position
                self.GenerateTable(s+1,a)

        return

#a recursive function used to
#generate the list of BCK-algebras for the next level,
#up to the specified level
#parameters:
#alg - the source algebra,
#level the desired level
    def GenerateNextLevel(self,alg,level=4):
#the dimension of the new algebras
        d=alg.n+1
        #check if the specified level is processed
        if d>level: return
#add a empty list for the current level
        self.nl.append([])
        pos=len(self.nl)-1
#construct a new algebra of order n+1
        a=LAlgebra(d,alg.den,alg)
#set the fixed elements on the last line
# and on the last column
        a.t[d-1][0]=d-1

        a.t[d-1][d-1]=0
#call the generate table function
#to find all algebras of order N+1
        self.GenerateTable(1,a)
#print the header of the current level
        print('\n{:5} {:d}\n{:4}\n'.format(
        'Level',d,'Results'))
        e=len(self.nl[pos])
#print all found algebras on the current level
#together with the BCK-algebras of the next level
#that are build from them
        for i in range(0,e):
            self.nl[pos][i].CheckBCKProperties()
            print('\n{:6d}'.format(i+1))
            print(self.nl[pos][i])
#autocall to generate next level BCK-algebras
```

```
            self.GenerateNextLevel(self.nl[pos][i],level)
```

Listing 3: main script

```
if __name__ == '__main__':
#import lalgebra class
    from LAlgebra import \ast
#import the timer library
    from timeit import default_timer as timer
#import the multiprocessing library
    import multiprocessing
#import the defs module
#this module contains a definition for a function
# which starts the build process of a BCK tree
#this separation was needed
# to be able to run multiprocessing
    import defs as d
#start the timer
    start = timer()
#define a function
#to start the process of BCK tree generation
#using multiprocessing
#How python has a strange multithreading behaviour,
#we used multiprocessing to optain true parallelization
#parameters:
#algl - the start BCK-algebras list,
#level - the desired level
    def mgen(algl,level):
#print the third level header
        print('\n{:5} {:d}\n{:4}\n'.format(
        'Level',3,'Results'))
#define a list of processes,
#each process will generate a BCK tree
        ltree=[]
        for i in range(0,3):
#print the current start BCK-algebra
            print('\n{:6d}'.format(i+1))
            print(algl[i])
#create a new process for the current BCK-algebra
# to generate the desired BCK tree
            thread=multiprocessing.Process(
            target=d.talg,args=(algl[i],level, ))
#add the process in the process list
            ltree.append(thread)
```

```
#start the process
            thread.start()

        for i in range(0,3):
#make the main process wait the other processes to finish
            ltree[i].join()

    #define a sequential execution function
# to start the generate process for BCK-algebras
#parameters:
#algl - the list of start BCK-algebras,
#level - the desired level
    def gen(algl, level):
#define a list of BCK-trees
        ltree=[]
#print the level 3 header
        print('\n{:5} {:d}\n{:4}\n'.format(
        'Level',3,'Results'))
        for i in range(0,3):
#print the current start BCK-algebra
            print('\n{:6d}'.format(i+1))
            print(algl[i])
        #call the function
#which generates a BCK tree
#for current BCK-algebra
            ltree.append(d.talg(algl[i],level))

    #build the list of BCK-algebras of order 3
    algl=[
    LAlgebra(3,'alg-1'),
    LAlgebra(3,'alg-2'),
    LAlgebra(3,'alg-3')]
#set the values for the first BCK-algebra of order 3
    algl[0].t[1][0]=1
    algl[0].t[2][0]=2
    algl[0].t[2][1]=1
#set the values for the second BCK-algebra of order 3
    algl[1].t[1][0]=1
    algl[1].t[2][0]=2
    algl[1].t[2][1]=2
#set the values for the third BCK-algebra of order 3
    algl[2].t[1][0]=1
    algl[2].t[1][2]=1
    algl[2].t[2][0]=2
```

```
    algl[2].t[2][1]=2
#build a list with the two functions variants,
#one for sequential executing, other for multiprocessing
    f=[gen,mgen]
#execute the sequential one
#to generate the BCK-algebras
#of order 4 and 5
#to call the multiprocessing function, change 0 with 1
    f[0](algl,5)
    #stop the timer
    end = timer()
#print the execution time
    print(
'\n Execution time: {} seconds'.format(end - start))
```

Listing 4: Defs script

```
#import the logical algebra class
from LAlgebra import \ast
#define the function
#which will start the BCK-tree generation
#for a specified algebra and a desired order
def talg(alg,level):
#define a bck tree
    tr=BCKTree(alg)
#call the GenerateNextLevel function
    tr.GenerateNextLevel(alg,level)
#return the generated BCK tree
    return tr
```

6. Conclusions

We have introduced in this paper a new way of building higher-order BCK-algebras starting from the lower-order ones, namely the generalized Iseki construction. We have shown that both extensions, the Iseki extension and the new Pseudo-Iseki extension are particular cases of the generalized Iseki construction. We pointed that the generalized Iseki construction can not be applied entirely to all BCK-algebras. Also, we have shown that it preserves the property of positive implicative. Moreover, we have shown that the generalized Iseki construction preserves commutativity only in its particular case, namely the Pseudo-Iseki extension. In further research, it will be interesting to see if such extensions can be used to reduce the necessary time for generating all BCK-algebras of order $\leq N$, where n is a given natural number. Also, we introduced new identities for positive implicative BCK-algebras, and we have shown an algorithm for gener-

ating BCK-algebras. This algorithm was used in finding and proving some of the identities shown in this paper. With further research, the algorithm can be improved through various techniques.

Acknowledgements

This work is supported by the project ANTREPRENORDOC, in the framework of Human Resources Development Operational Programme 2014-2020, financed from the European Social Fund under the contract number 36355/23.05.2019 HRD OP /380/6/13 – SMIS Code: 123847.

References

[1] Arsham Borumand Saeid, Cristina Flaut, Sarka Hoskova-Mayerova, Radu Vasile, *Wajsberg algebras of order n, $n \leq 9$*, Neural Comput Appl., 32 (2020), 13301-13312.

[2] Sarka Hoskova-Mayerova, Cristina Flaut and Radu Vasile, *Some remarks regarding finite bounded commutative BCK-algebras*, Algorithms as a foundation for modern applied Mathematics (Cristina Flaut, ed.), Studies in Fuzziness and Soft Computing, Springer, 2020.

[3] Wieslaw Dudek, *Remarks on the axioms system for bci-algebras*, Prace Naukowe WSP w Czestochowie, 2 (1996), 46-61.

[4] C. Flaut, *Bck-algebras arising from block-codes*, J. Intell. Fuzzy Syst., 4 (2015), 1829-1833.

[5] Cristina Flaut, Radu Vasile, *Wajsberg algebras arising from binary block-codes*, Soft. Comput., 24 (2020), 6047-6058.

[6] Yasuyuki Imai, Kiyoshi Iséki, *On axiom systems of propositional calculi, xiv*, P. Jpn. Acad., 42 (1966).

[7] Y. B. Jun, S. Z. Song, *Codes based on BCK-algebras*, Inform Sciences, 11 (2011), 5102-5109.

[8] Antonio Maturo, Šárka Hošková-Mayerová, Daniela-Tatiana Soitu, and Janusz Kacprzyk (eds.), *Recent trends in social systems: Quantitative theories and quantitative models*, Studies in Systems, Decision and Control, vol. 66, ch. Some Connections Between Binary Block Codes and Hilbert Algebras, pp. 249-256, Springer International Publishing Switzerland, 2017.

[9] J. Meng, Y. B. Jun, *Bck-algebras*, Kyung. Moon. Sa Co., Seoul, 1994.

[10] Sambasiva Rao Mukkamala, *A course in be-algebras*, Springer Nature Singapore Pte Ltd., Singapore, 2018.

[11] Radu Vasile, *New ways for building BCK-algebras of higher order*, Tbil. Math. J., 13 (2020), no. Special Issue iecmsa-2019, 111-124.

Accepted: June 3, 2021