Article

Using Extended Logical Primitives for Efficient BDD Building

David Fernandez-Amoros , Sergio Bra, Ernesto Aranda-Escolástico and Ruben Heradio

Deptment of Computer Systems and Software Engineering, Universidad Nacional de Educacion a Distancia (UNED), 28040 Madrid, Spain; sbra1@alumno.uned.es (S.B.); earandae@issi.uned.es (E.A.-E.); rheradio@issi.uned.es (R.H.)

* Correspondence: david@issi.uned.es

Received: 5 July 2020; Accepted: 29 July 2020; Published: 31 July 2020

Abstract: Binary Decision Diagrams (BDDs) have been used to represent logic models in a variety of research contexts, such as software product lines, circuit testing, and plasma confinement, among others. Although BDDs have proven to be very useful, the main problem with this technique is that synthesizing BDDs can be a frustratingly slow or even unsuccessful process, due to its heuristic nature. We present an extension of propositional logic to tackle one recurring phenomenon in logic modeling, namely groups of variables related by an exclusive-or relationship, and also consider two other extensions: one in which at least n variables in a group are true and another one for in which at most n variables are true. We add XOR, atLeast- n and atMost- n primitives to logic formulas in order to reduce the size of the input and also present algorithms to efficiently incorporate these constructions into the building of BDDs. We prove, among other results, that the number of nodes created during the process for XOR groups is reduced from quadratic to linear for the affected clauses. the XOR primitive is tested against eight logical models, two from industry and six from Kconfig-based open-source projects. Results range from no negative effects in models without XOR relations to performance gains well into two orders of magnitude on models with an abundance of this kind of relationship.

Keywords: binary decision diagrams; software product lines; circuit testing; configurable systems

1. Introduction

Formal methods use logical representations of models to enable complex analyses. One such representation are Binary Decision Diagrams (BDDs), an evolution of Binary Decision Trees [1]. BDDs can represent the solutions of a propositional logic formula in a way which allows to easily compute properties of the solutions. In this regard, BDDs are a competing technology to SAT-solvers and #SAT-solvers. A SAT-solver is a program that takes a propositional formula and determines if it is satisfiable. It does so by running a heuristic tree search over the variables in the formula. #SAT-solvers go further and compute the number of satisfying assignments (i.e., the number of solutions to the problem). SAT and #SAT-solvers have enjoyed much attention in the past years due to the so-called SAT revolution during which great progress (of several orders of magnitude) was made in terms of performance.

A BDD compiles the information about the formula in a graph that can be traversed to compute sophisticated analyses of the model. The advantage of BDDs is that, for many problems, a single traversal of the graph can produce the solution, while SAT-solver technology often requires creating many slightly different formulas and calling the solver for each one of the variants, with the subsequent loss of efficiency. Another interesting property of BDDs is that its construction does not require the input to be in Conjunctive Normal Form (CNF) as opposed to SAT-solvers. Conversion to CNF can produce an exponentially longer formula, which is a bad start for an NP-complete problem such as SAT [2].

The main disadvantage of BDDs is that the construction of the graph, also known as BDD synthesis, may explode in terms of time and space, and finding an optimal graph is also an NP-complete problem [3]. There are several variations of BDDs, but the most widely employed is Reduced Ordered Binary Decision Diagrams (ROBDDs), to the point that they are usually just called BDDs. These BDDs are ordered in the sense that there is an ordering of the variables that cannot be violated (i.e., for each parent and child nodes, the variable of the parent appears earlier in the ordering than the variable of the child). BDDs are known to be very sensitive to the ordering of the variables, so that different orders can produce BDDs with very different numbers of nodes. Just as SAT-solvers rely on heuristic search, BDD synthesis relies on heuristics to find a good ordering. So far, there has been no BDD revolution, so interest in BDDs has been declining.

This paper hopes to spur back interest in BDDs by providing a performance improvement of several orders of magnitude for some models. Instead of delving into the waters of heuristic orderings, our aim is to exploit common structures in logical models, namely *atLeast-n*, *atMost-n* and XOR constraints over variables $x_1, x_2, \dots, x_m$ meaning that at least n variables are true, at most n variables are true and that exactly one of the variables has to be true, respectively. Instead of translating these constructions to propositional logic, which would produce a combinatorial explosion of clauses (a clause is a disjunction of literals. A literal is a variable v or its negation $\neg v$), our approach consists in creating new primitives; *atLeast-n*, *atMost-n* and XOR, thereby extending our logic representation.

The standard way of synthesizing BDDs applies one clause at a time to the BDD. We will prove that for the encoding of *atLeast*, *atMost* and XOR, this creates at least one new node for each clause, many of which are not used in the final BDD. Apart from speeding up the translation, our approach can be directly implemented in the graph, meaning that only the necessary nodes are created. In the case of the XOR, this means a reduction in the creation of nodes from quadratic to linear for the parts of the model which display this pattern. We will look for the XOR constructions in two different realms; Feature models and the translation to logic of Kconfig language configuration files, and show that our approach builds BDDs significantly faster (up to two orders of magnitude) when XOR groups are identified and imposes no penalty if there are none.

Since BDD synthesis often relies on heuristics and different parameters, speeding up the process is not only useful for pursuing a specific application, but also for the advancing of the systematic knowledge about which techniques work best in particular scenarios, as it allows to try more approaches in the same amount of time.

The remainder of the paper is structured as follows: Section 2 covers related work. Section 3 deals with translating some FM notations to propositional logic. Section 4 shows how to extend a BDD engine to build XOR groups efficiently and proves some lower bounds on the number of nodes created. Section 5 explains the experimental evaluation performed and Section 6 discusses the conclusions and future work.

2. Related Work

One field in which both SAT-solvers and BDDs have been employed is Software Product Lines (SPLs). A SPL is an engineering effort to produce families of related software products reusing assets, also known as features. SPLs are often described using Feature Models (FMs) [4]. Figure 1 shows the FM for a smart home. A FM uses a predetermined notation to represent a product line. Mandatory nodes represent features which are always present. Optional nodes can be present or not in a product. Any of the children of an or-node can be selected. Finally, exactly one of the children of a xor-node can be present in a product. Feature models can also depict constraints among the features, in this case, there are three of them:

- *Viewer* requires *Reflective Data Structure*
- *Execute Tests* requires *Deploy Tests*
- *Authorization* requires *Authentication*

Figure 1. A feature model for a Smart Home system.

These relationships have been graphically shown as arrows in the diagram. The XOR and OR constructions in the feature model match exactly the logical XOR and atLeast-1 extensions, respectively. For example, the *Test* subtree has two OR descendants, which is equivalent to atLeast-1(*Deploy Tests*, *Execute Tests*). Furthermore, *Execute Tests* requires *Deploy Tests*. Therefore, there are only three valid configurations for this subtree:

1. $\{\neg \text{Test}, \neg \text{Deploy Tests}, \neg \text{Execute Tests}\}$
2. $\{\text{Test}, \text{Deploy Tests}, \neg \text{Execute Tests}\}$
3. $\{\text{Test}, \text{Deploy Tests}, \text{Execute Tests}\}$

It is worth noting that Feature Oriented Display Analysis (FODA) [4] supports representing the same semantics in different graphical ways, depending on the modeler's intentionality. For example, an interchangeable *Test subtree* would have *Deploy Tests* as mandatory and *Execute Tests* as optional.

Product line analysis of a FM can be used to determine a great deal of properties about the product line, for instance features that are never used (i.e., dead features) [4–6], or features that are always used (i.e., core features) [5,6], as well as the probability of any particular feature being present in a product [7].

So far, product line analyses have been restricted to small models, simplified models, superficial analyses or a combination thereof [8–15] due to the computational complexity of dealing with formulas involving large numbers of variables with complex relations between them, as well as the need to perform a large number of calls to the logic engine of choice.

Suppose, for instance, that you need to know the probability of a particular feature in a product line. This can be useful to create a random sample of products, as well as to assess the uniformity of the sampling approach [16]. A common approach is to call a model counter once to get the total number of products of the model ϕ , and then once again for each feature f , with the argument $\phi \wedge f$, and divide the latter by the former to obtain the probability of f .

However efficient model counters are, this involves repeating a great deal of the heuristic search. For this reason, it is much more efficient to use BDDs to compute all the probabilities in one traversal of the graph [7]. However, synthesizing the BDDs can be a resource-consuming task.

BDD synthesis has traditionally been a complicated issue in which, lacking a general approach, each underlying domain has relied upon different techniques. BDDs have been used to represent fault trees in plasma reactors [17–20] perform circuit optimization and verification [21–34] and in software product lines [6,7,11,15,35–45]. The circuit domain can sometimes add additional hypotheses, such

as gates, which are represented by variables that occur only once in the model formula to obtain better bounds or even optimal algorithms. Nevertheless, some ordering heuristics, both static and dynamic, have enjoyed the benefit of being useful across the board, such as FORCE [46], MINCE (Min Cut Etc.) [47] among the static ones and sifting [48,49] as a dynamic reordering heuristic.

The extension to logic that we propose in this paper could easily fit with small changes into what is known in the SAT community as Satisfiability Modulo Theories (SMTs) that is, using standard SAT techniques together with a black box that solves particular problems. Some examples of SMTs include the use of uninterpreted functions, different algebraic extensions to logic, and even XOR clauses, although this applies to bitwise XOR, i.e., checking if the number of 1s is odd, as often used in hashing and cryptography, not to the kind of cardinality constraint we have been discussing in this paper. However, this black-box approach is supposed to incur in a performance penalty. A more interesting approach would be to create a hybrid SAT, in this case, a conventional SAT solver that integrates XOR restrictions as a part of the Boolean Constraint Propagation (BCP) engine.

In any case, adapting the approach to SAT technology is outside the scope of this paper. There is compelling evidence that BDDs perform better than SAT when the BDD synthesis is possible, at least for some SPL engineering problems [6,7,50–52]. This is especially true w.r.t. #SAT solvers, which need to always traverse the whole search space.

Cardinality constraints are common in variability models. Figure 2 summarizes the percentage of XOR, atLeast-1 and atMost-1 constraints that Berger et al. [53] found after analyzing 128 variability models from 12 open-source projects specified with the languages SPLOT, CDL, and Kconfig. In general, the atLeast and atMost operators are expected to be found less commonly in logical models. However, the XOR construction is very common because, as Biere et al. [54] explain, it is the natural way to encode a variable, v , taking values in a discrete domain of values $\{v_1, v_2, \dots, v_n\}$. Another reason is that XOR is already the conjunction of atLeast-1 and atMost-1.

Figure 2. Percentage of grouping constraints that Berger et al. [53] found in 128 real variability models.

In this paper we associate high-level language constructions to patterns. An alternative approach is to look for the XOR, atLeast and atMost operators directly into the Boolean formula [54].

3. Translating FMs to Extended Propositional Logic

Feature models are an interesting application domain for testing the effectiveness of the approach. In this section, we apply the extended propositional logic to two important sources of FMs, namely FODA and Kconfig models.

3.1. FODA

FODA stands for Feature Oriented Domain Analysis [4], and it provides a notation to represent FMs graphically. In FODA, a model is represented by a tree of features with different relations between them. An example FODA model can be seen in Figure 1. We are interested here mostly in the XOR group cardinality, in which only one feature in a group is to be present in the product. In the example in Figure 3 we can see one of these XOR groups applied to the different ways of taking a screenshot in Ubuntu. FODA models can also incorporate cross-tree constraints (i.e., arbitrary logical constraints) between the features to further restrict the products in the product line.

Figure 3. A Ubuntu screenshot FM.

The translation of FODA models to standard propositional logic is rather straightforward [37]. It can be summarized in the following steps:

1. Every feature f , except the root feature, has a parent feature p , and the child must imply the parent: $f \rightarrow p$
2. in the case of a mandatory feature, f , the parent, p , also implies the child: $p \rightarrow f$
3. in the case of an Or group of n features, at least one of the features is selected. For $f_1, f_2, \dots, f_n$: $\bigvee_{i=1}^n f_i$
4. in the case of an XOR group of n features, exactly one of the features is true and the others are false. For $f_1, f_2, \dots, f_n$: $\bigwedge_{i < j} \neg(f_i \wedge f_j) \wedge \bigvee_{i=1}^n f_i$
5. The cross-tree constraints need not be translated, as they are already in logical format.

The problem with this translation lies in step 4. The disjunction of the features does not pose a problem, but the part of not more than one feature being true is more problematic, as the number of clauses grows very fast. For n features, this would translate in $n(n-1)/2$ clauses in the translation, a quadratic growth. It is not uncommon to find XOR groups with hundreds of features and in these cases the clauses related to XOR groups make up the vast majority of the translation. We propose to extend our set of logical primitives with a new one: XOR. This way, step 4 of the translation would simply read like this:

In the case of an XOR group of n features: $f_1, f_2, \dots, f_n$, the translation would be: $\text{XOR}(f_1, f_2, \dots, f_n)$.

3.2. Kconfig

Figure 4 shows a snippet of code in the Kconfig language, which is used to configure the Linux kernel, among other open source projects. The human configurator needs to answer a series of questions in order to assign values to a set of symbols which are used to configure the kernel before it is compiled. The language allows different value types for the symbols, namely bool, tristate, string, int and hex. The vast majority of symbols for the evaluated projects are bool, so we only translate those for simplicity. The choice construction in Kconfig means that the user must select exactly one of the options. In this case, some buffer needs to be allocated in memory and the user is given three choices as to the location. The kconfig language is complicated but its translation to logic has been extensively researched [55–61]. Suffice it to say that the meaning of the choice construct for Boolean symbols in Kconfig perfectly matches the XOR construction (i.e., one the options is true and the rest are false).

```
choice
prompt "Buffer allocation policy"
default GO_ON_STACK
config USE_MALLOC
    bool "Allocate with malloc"
config GO_ON_STACK
    bool "Allocate on the stack"
config GO_IN_BSS
    bool "Allocate in the .bss section"
endchoice
```

Figure 4. A Kconfig code snippet.

4. Building BDDs for Extended Logic

Our approach analyzes the model to identify parts equivalent to XOR, atLeast, and atMost. Then, it translates those parts into the extended logic constructs. This way, the size of the Boolean formula is reduced. Then, the new primitives in the formula are exploited to build the BDD more efficiently. For instance, consider again the code snippet in Figure 4. The translation to logic, which uses standard propositional logic, can be seen in Figure 5, while the translation using the XOR primitive is showed in Figure 6.

Figure 7 shows the BDD corresponding to an XOR primitive of variables $x_1, x_2, \dots, x_n$. The edges going to node 0 and node 0 itself have been omitted for clarity. The solid lines represent setting the variable to true and connect the parent with the high child, and the dotted edges mean setting it to false, and connect the parent with the low child. This BDD only has $2n - 1$ nonterminal nodes. The novelty of this approach is that the BDD can be created using exactly one call to the BDD engine per node. The pseudocode needed to generate this BDD is shown in Algorithm 1. The BDD graph is built in a bottom-up fashion. To do this, the list of variables must be first ordered in descending position order, so that the variables corresponding to the lower levels in the graph go first, and the variable in the top of the BDD is the last one treated. The algorithm relies on a function *CreateOrFind*(*var*, *thenNode*, *elseNode*), which looks for a node corresponding to variable *var*, with children *thenNode* and *elseNode*. If such a node exists, it is returned, if it does not, it is created and returned. Luckily this function is already known in efficient BDD engine programming as it is the basis to build the unique table [62].

```
USE_MALLOC or GO_ON_STACK or GO_IN_BSS
not USE_MALLOC or not GO_ON_STACK
not USE_MALLOC or not GO_IN_BSS
not GO_ON_STACK or not GO_IN_BSS
```

Figure 5. A translation to propositional logic.

```
XOR(USE_MALLOC, GO_ON_STACK, GO_IN_BSS)
```

Figure 6. A translation using the extended XOR primitive.

Figure 7. A Binary Decision Diagram (BDD) to represent $\text{XOR}(x_1, x_2, \dots, x_n)$.

Algorithm 1: Direct construction of the BDD for an XOR primitive

Data: L - a list of variables
Result: root - a BDD subgraph representing the XOR construction
 $L \leftarrow$ sort L in descending variable order
 $\text{lastvar} \leftarrow$ variable with the lowest index
 $\text{prevTrueCol} \leftarrow 1$
 $\text{prevFalseCol} \leftarrow 0$
forall $\text{var} \in L - \{\text{lastvar}\}$ **do**
 $\text{prevTrueCol} \leftarrow \text{FindOrCreate}(\text{var}, 0, \text{prevTrueCol})$
 $\text{prevFalseCol} \leftarrow \text{FindOrCreate}(\text{var}, \text{prevTrueCol}, \text{prevFalseCol})$
end
 $\text{root} \leftarrow \text{FindOrCreate}(\text{lastvar}, \text{prevTrueCol}, \text{prevFalseCol})$
return root

Figure 8 shows the BDD for the atLeast-3 relation. The algorithm to compute the BDD for atLeast is slightly more complex than the XOR case and can be found in Algorithm 2.

Figure 8. A BDD representing atLeast-3 for $m > 3$.

Algorithm 2: Direct construction of the BDD for atLeast – n primitive

Data: L - a list of literals of size m
Data: n : integer
Result: root - a BDD subgraph representing the atLeast construction
 $L \leftarrow \text{sort } L \text{ in descending variable order}$
 savedNodes : array of nodes
if $n > m$ **then**
 return 0 ;
if $n = m$ **then**
 lastNode $\leftarrow$ 1;
 forall lit in L **do**
 var $\leftarrow$ getVar(lit);
 if lit is var **then**
 lastNode $\leftarrow$ FindOrCreate(var, lastNode, 0)
 else
 lastNode $\leftarrow$ FindOrCreate(var, 0, lastNode)
 return (lastNode) ;
 end
for column $\leftarrow n$ **to** 1 **by** -1 **do**
 for lit $\leftarrow m-n+column$ **to** column **by** -1 **do**
 var $\leftarrow$ getVar(lit);
 if column = n **then**
 childThen $\leftarrow$ 1 ;
 else
 childThen $\leftarrow$ savedNodes[column+1][var+1] ;
 if var = $m-n+column$ **then**
 childElse $\leftarrow$ 0 ;
 else
 childElse $\leftarrow$ savedNodes[column][var+1];
 if lit is var **then**
 savedNodes[column][var] = FindOrCreate(var, childThen, childElse) ;
 else
 savedNodes[column][var] = FindOrCreate(var, childElse, childThen) ;
 end
end
return (savedNodes[1][1]) ;

Computing the BDD for atMost relations can be reduced to the atLeast case because $\text{atMost-}n(x_1, x_2, \dots, x_m) \equiv \text{atLeast-}(m-n)(\neg x_1, \neg x_2, \dots, \neg x_m)$. Algorithm 2 works with literals, so it can be used as is to compute this last expression.

We prove later in this section that the usual encodings of XOR, atLeast- n and atMost- n produce $1 + \binom{m}{2}$, $\binom{m}{n-1}$ and $\binom{m}{n+1}$ clauses, respectively, that have to be applied to the BDD iteratively, while the extended approach reduces this to a single clause. The final result is computed directly in the extended approach with node creation reduced to the strictly minimal, while the traditional one creates intermediary nodes that are later deleted as clause processing continues.

To exemplify this behavior, consider the formula $\text{XOR}(a, b, c)$. In traditional logic, the encoding has four clauses: $a \vee b \vee c$, $\neg a \vee \neg b$, $\neg a \vee \neg c$, $\neg b \vee \neg c$. Figure 9, shows how the BDD changes as each clause is applied. The orange nodes are nodes that have been created in that step. In Figure 9a, a node for each variable is created. In Figure 9b, a new node for b is created. Because nodes are created and identified by the variable they represent and its two children, the node for a has also been created

because its high child has changed from 0 to the new node for b . In Figure 9c, there is a new node for c , which in turn needs a new node for b which prompts for a new node for a . In the last step, a new node for b is created to account for the change in its high child, which prompts the creation of a new node for a .

Figure 9. Traditional building of $\text{XOR}(a, b, c)$. (a) Applying $a \vee b \vee c$. (b) Applying $\neg a \vee \neg b$. (c) Applying $\neg a \vee \neg c$. (d) Applying $\neg b \vee \neg c$.

In total, during four steps, ten nodes have been created, of which only five were necessary. The extended logic approach involves one step, creates five nodes and does not create any unnecessary nodes. The traditional encoding is inefficient because it creates a quadratic number of nodes, more than $1 + \binom{n}{2}$, to retain only $2n - 1$ at the end. The extended approach computes only the necessary nodes. We now establish a lower bound for the number of nodes created using the traditional encoding. The bound is not tight because different orderings of the clauses produce different amounts of intermediary nodes, even if the final BDD is the same.

Definition 1. A node in a BDD is either the terminal node 0, the terminal node 1, which have no children, or it is a nonterminal node uniquely determined by the variable it represents together with two children nodes called low and high, which represent the effect of setting the variable to false and to true, respectively.

BDDs are reduced, which means there can be no two different nodes with the same variable and children so, the previous definition makes sense. The following lemma provides a lower bound to the creation of new nodes in a series of BDDs which mimicks the repeated use of the apply operation to create a BDD representing a set of clauses. Please note that we are interested in the creation of new nodes, not in the total number of nodes in the BDD, which may grow or shrink during the apply process, because a node created can be later discarded when it is no longer part of the BDD.

Lemma 1. Let $V = \{x_1, x_2, \dots, x_m\}$ be a set of variables, $C = \{c_1, c_2, \dots, c_n\}$ a set of clauses over V , such that $\bigwedge_{j=1}^{i-1} c_j \not\models c_i$, and its conjunction is satisfiable, and let $B = \{b_0, b_1, \dots, b_n\}$ be a set of BDDs over V such that b_0 represents the formula $\top$ and b_i with $i > 0$ represents the formula $\bigwedge_{j=1}^i c_j$. Then, each BDD b_i , $1 \leq i \leq n$ has at least a new node w.r.t its predecessor.

Proof. The proof uses induction over the succession of BDDs b_i when $i > 0$: b_0 has no nonterminal nodes. b_1 represents c_1 which is satisfiable and cannot be $\top$ or $\perp$, so b_1 has at least one nonterminal node. We now assume that the lemma holds for all b_j , $1 \leq j \leq k$ and proceed to prove that b_{k+1} has at least one new node w.r.t. its predecessor. By the induction hypothesis we know that at least one new node is created at each apply step since the first, so b_k has at least one nonterminal node, and so, there is a nonterminal node as the root. The change from b_k to b_{k+1} consists of applying c_{k+1} to b_k . Since, by hypothesis, $\bigwedge_{j=1}^k c_j \not\models c_{k+1}$, it means that the set of satisfying assignments for b_{k+1} is a strict subset than that of b_k . Let σ be one variable assignment that satisfies b_k but not b_{k+1} . In b_k there is one path from the root to node 1 which contains σ . b_{k+1} also has a nonterminal root, since it represents neither $\top$ nor $\perp$. Because σ does not satisfy b_{k+1} , there is one path from the root of b_{k+1} to 0 which contains σ . At least one of the nodes along this path is new w.r.t. b_k , otherwise they are present also in b_k and that would contradict σ satisfying b_k . $\square$

Now we use the lemma to prove three theorems about the number of nodes created by the traditional encoding of the cardinality operators.

Theorem 1. The traditional encoding to create a BDD for $\text{atMost-}m(x_1, x_2, \dots, x_m)$ creates at least $\binom{m}{n+1}$ nodes.

Proof. The encoding for $\text{atMost-}n$ has $\binom{m}{n+1}$ clauses, namely,

$$C = \{\neg y_1 \vee \neg y_2 \vee \dots \vee \neg y_{n+1} \mid y_k \in V, 1 \leq k \leq n+1, y_i \neq y_j \text{ if } i \neq j\}$$

Let σ be a variable assignment with at most n variables set to true and let c be a clause in C . C has $n+1$ variables so at least one of them is assigned false in σ , which means that σ satisfies c . Now let M be a model for C and let τ be a variable assignment from M . At most n variables in τ are true, because if there are $n+1$ variables assigned to true, there is one clause in C which is not satisfied, which contradicts M being a model of C .

The conditions of the Lemma 1 apply to C , whichever the order of the clauses, so at least $\binom{m}{n+1}$ nodes are created. $\square$

Theorem 2. The traditional encoding to create a BDD for $\text{atLeast-}n(x_1, x_2, \dots, x_m)$ creates at least $\binom{m}{n-1}$ nodes.

Proof. The encoding for $\text{atLeast-}n$ has $\binom{m}{n-1}$ clauses, which is equal to $\binom{m}{m-n+1}$, namely,

$$C = \{y_1 \vee y_2 \vee \dots \vee y_{m-n+1} \mid y_k \in V, 1 \leq k \leq m-n+1, y_i \neq y_j \text{ if } i \neq j\}$$

Let σ a variable assignment with at least n variables assigned to true. That means that at most $m-n$ variables are set to false, so if we choose any group of $m-n+1$ variables, at least one is true, so all the clauses in C are satisfied by σ . Now let M be a model of C and τ a variable assignment from τ . At least n variables from τ must be assigned to true, otherwise there is a group of $m-n+1$ variables in τ which are assigned to false, and then one clause in C would not be satisfied.

Again, the conditions of Lemma 1 are met regardless of the order of the clauses, so at least $\binom{m}{n-1}$ nodes are created. $\square$

Theorem 3. The traditional encoding to create a BDD for $\text{XOR}(x_1, x_2, \dots, x_m)$ creates at least $1 + \binom{m}{2}$ nodes.

Proof. The traditional encoding has $1 + \binom{m}{2}$ clauses, namely,

$$C = \{x_1 \vee x_2 \dots \vee x_m\} \cup \{\neg x_i \vee \neg x_j \mid y_i \neq y_j \text{ if } i \neq j\}$$

The first part corresponds to atLeast-1 and the second to atMost-1, according to the two previous theorems. These clauses can be applied in any order, but they meet the conditions of Lemma 1. Since there are $1 + \binom{m}{2}$ steps, there at least as many nodes created along the process. $\square$

5. Experimental Validation

5.1. Research Questions

Our experimental evaluation targets three research questions:

- **RQ1: Boolean formula size-reduction.** Does extended logic decrease the Boolean formula number of clauses in practice?
- **RQ2: Time for obtaining extended logic formulas.** How much time does it take to translate an input model into a Boolean formula with extended logic compared to the non-extended logic translation?
- **RQ3: BDD synthesis time.** Does extended logic reduce BDD build time in practice?

5.2. Experimental Setup

For the evaluation, we compared the building process of BDDs with and without using the XOR connective for a number of software projects. We did not find any occurrences of atLeast-n or atMost-n with n bigger than one. All occurrences of atMost-1 and atLeast-1 occurred together, so we considered them as XOR. Table 1 shows the systems used as test-bed. The systems chosen vary in size from very small to very large, so the scalability of the approach may be adequately assessed. Furthermore, the systems were drawn from two different sources, namely open-source projects using the Kconfig configuration system, and two feature models from industry using the SPLOT format.

Table 1. Number of features and XOR-groups for each system.

System	#Features	#XOR-Groups
Toybox	10	0
axTLS	64	6
DellSPLOT	118	13
Fiasco	122	18
μ Clibc	303	27
BusyBox	604	8
EmbToolKit	2325	130
Automotive02	17,365	1161

The SPLOT repository (<http://www.splot-research.org>) contains over a thousand feature models in an XML format that has become a standard for FODA-like projects. This splot file format is widely used elsewhere, so we developed the tool *splot2model* that translates a splot file to two files, one to contain the order of variables and another one to hold the formulas. This tool has an option to use the XOR connective or not. We chose an automotive model from [63] and a Dell laptop model from [64]. For Kconfig files, we developed a similar approach, with a tool called *Kconfig2Logic* with an option to use the XOR construction.

All the tools and data needed to replicate the experiments can be found in: <https://github.com/davidfa71/Extending-Logic>.

The BDD engine used was the CUDD package (<https://github.com/vscosta/cudd>), for which we coded the extension discussed in the previous section. The FindOrCreate function is already builtin

in CUDD and is called UniqueTable. The experiments were run in an HP Proliant Server with 2 Xeon E5-2660v4 CPUs (2.0 Ghz).

5.3. Results

5.3.1. Boolean Formula Size-Reduction (RQ1)

Each logical model is the conjunction of a number of clauses. We compared the number of clauses produced in both of the translations of the projects. The results can be seen in Figure 10.

There is a significant reduction in the number of extended clauses, except for Toybox, which does not include any XOR-group at all, and BusyBox, which has a very low proportion of XOR-groups related to its number of features. The most extreme cases are Fiasco and Automotive02. For Fiasco, the standard translation has 2364 clauses, and the extended translation has 186 clauses, a reduction of 92.14%. For Automotive02, the translations have 321,933 and 5616 clauses respectively, a reduction of 99.98%.

Figure 10. Number of (extended) clauses for standard and extended logic.

5.3.2. Time for Obtaining Extended Logic Formulas (RQ2)

The next question was to assess if the extended logic imposes a time penalty when translating the model in its original form (Kconfig code or SPLIT format) to logic. Figure 11 shows the results.

Figure 11. Translation times to standard and extended logic translation.

The extended logic translation was faster in all cases, except for Toybox for which both translations are the same. For the biggest models, the translation was much faster with the extended logic, notoriously for Fiasco and Automotive02, for which the extended approach was seven and sixteen times faster, respectively, which indicates that performance gains correlate well with the number of XOR constraints relative to the number of features.

5.3.3. BDD Synthesis Time (RQ3)

The last research question, which is the most important one, refers to whether extended logic helps to speed up BDD construction. Figure 12 shows the results.

The build times for the four smaller models are very similar for both approaches, but for the four biggest (the second row in the plot) the performance gains are considerable: for Fiasco, building the BDD with the extended logic was almost 10 times faster, and for Automotive02, it was 205 times faster than using the standard logic translation, from 1 h and 20 min to 23 sec, so the answer is yes, especially if the model contains a lot of exclusive-or cardinality restrictions.

Figure 12. BDD build times for standard and extended logic translations.

6. Conclusions and Future Work

Efficient building of BDDs is of paramount importance to its widespread use. A long building phase can easily offset the benefits of fast analysis algorithms. In this paper, we have identified a major obstacle in BDD building and developed a mitigation strategy with a complexity reduction from a quadratic number of created nodes to a linear one for the occurrences of the XOR primitive, which can amount to a sizable amount of the translation of a model. Our experimental evaluation has confirmed the theoretical gains with an improvement of building times of up to two orders of magnitude for real-life models, which indicates that BDDs have still a lot of untapped potential. We also considered two other primitives, atLeast-n and atMost-n, for which we proved good theoretical properties, but found no real models that used them. An interesting extension would be to use the XOR implementation to simplify the translation of string, int, and hex values in Kconfig files, which suffer from exactly the same problems as described for the choice construction.

Author Contributions: Conceptualization: D.F.-A., S.B. and R.H.; data curation: D.F.-A. and S.B.; formal analysis: D.F.-A., S.B., R.H. and E.A.-E.; funding acquisition: R.H.; investigation: D.F.-A., S.B., R.H. and E.A.-E.; methodology: D.F.-A., S.B. and E.A.-E.; project administration: R.H.; resources: D.F.-A. and R.H.; software: D.F.-A., S.B., E.A.-E. and R.H.; supervision: D.F.-A. and R.H.; validation: D.F.-A. and S.B.; visualization: D.F.-A. and R.H.; writing—original draft: D.F.-A. and S.B.; writing—review & editing: E.A.-E. and R.H. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been supported by (i) the Spanish Ministry of Science, Innovation and Universities, under grant with reference DPI2016-77677-P, and (ii) the Community of Madrid, under the research network CAM RoboCity2030 S2013/MIT-2748.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Bryant, R.E. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. Comput.* **1986**, *35*, 677–691. [\[CrossRef\]](#)
2. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; ACM: New York, NY, USA, 1971; pp. 151–158.
3. Bollig, B.; Wegener, I. Improving the Variable Ordering of OBDDs is NP-Complete. *IEEE Trans. Comput.* **1996**, *45*, 993–1002. [\[CrossRef\]](#)
4. Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, S. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*; Technical Report CMU/SEI-90-TR-21; Software Engineering Institute: Pittsburgh, PA, USA, 1990.
5. Mendonça, M.; Wasowski, A.; Czarnecki, K. SAT-based Analysis of Feature Models is Easy. In Proceedings of the 13th International Software Product Line Conference, San Francisco, CA, USA, 24–28 August 2009; pp. 231–240.
6. Perez-Morago, H.; Heradio, R.; Fernandez-Amoros, D.; Bean, R.; Cerrada, C. Efficient Identification of Core and Dead Features in Variability Models. *IEEE Access* **2015**, *3*, 2333–2340. [\[CrossRef\]](#)
7. Heradio, R.; Fernandez-Amoros, D.; Mayr-Dorn, C.; Egyed, A. Supporting the Statistical Analysis of Feature Models. In Proceedings of the IEEE/ACM 41st International Conference on Software Engineering, Montreal, QC, Canada, 25–31 May 2019; pp. 843, 853.
8. Zaid, L.A.; Kleinermann, F.; De Troyer, O. Applying Semantic Web Technology to Feature Modeling. In Proceedings of the 2009 ACM Symposium on Applied Computing, Honolulu, HI, USA, 9–12 March 2009; ACM: New York, NY, USA, 2009; pp. 1252–256.
9. Bachmeyer, R.C.; Delugach, H.S. A conceptual graph approach to feature modeling. In Proceedings of the International Conference on Conceptual Structures, Sheffield, UK, 22–27 July 2007; Springer: Berlin, Germany, 2007; pp. 179–191.
10. Benavides, D.; Trinidad, P.; Cortés, A.R. Using Constraint Programming to Reason on Feature Models. In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, SEKE, Taipei, Taiwan, 14–16 July 2005; pp. 677–682.
11. Fernandez-Amoros, D.; Heradio, R.; Cerrada, J.A. Inferring Information from Feature Diagrams to Product Line Economic Models. In Proceedings of the 13th International Software Product Line Conference, San Francisco, CA, USA, 24–28 August 2009; Carnegie Mellon University: Pittsburgh, PA, USA, 2009.
12. Hemakumar, A. Finding Contradictions in Feature Models. In Proceedings of the Software Product Lines, 12th International Conference, SPLC, Limerick, Ireland, 8–12 September 2008; Second Volume (Workshops), pp. 183–190.
13. Kastner, C.; Apel, S.; Batory, D. A Case Study Implementing Features Using AspectJ. In Proceedings of the 11th International Software Product Line Conference (SPLC 2007), Tokyo, Japan, 10–14 January 2007; pp. 23–232. [\[CrossRef\]](#)
14. Mannion, M. Using First-Order Logic for Product Line Model Validation. In Proceedings of the Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, 19–22 August 2002; pp. 176–187. [\[CrossRef\]](#)
15. Mendonça, M.; Wasowski, A.; Czarnecki, K.; Cowan, D.D. Efficient compilation techniques for large scale feature models. In Proceedings of the Generative Programming and Component Engineering, 7th International Conference, GPCE, Nashville, TN, USA, 19–23 October 2008; pp. 13–22. [\[CrossRef\]](#)
16. Plazar, Q.; Acher, M.; Perrouin, G.; Devroey, X.; Cordy, M. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet? In Proceedings of the Validation and Verification (ICST), 12th IEEE Conference Software Testing, San Francisco, CA, USA, 4–9 April 2019; pp. 240–251. [\[CrossRef\]](#)
17. Bartlett, L.M.; Andrews, J.D. Choosing a heuristic for the "fault tree to binary decision diagram" conversion, using neural networks. *IEEE Trans. Reliab.* **2002**, *51*, 344–349. [\[CrossRef\]](#)

18. Jung, W.S.; Han, S.H.; Ha, J. A fast BDD algorithm for large coherent fault trees analysis. *Reliab. Eng. Syst. Saf.* **2004**, *83*, 369–374. [\[CrossRef\]](#)
19. Remenyte, R.; Andrews, J.D. A Simple Component Connection Approach for Fault Tree Conversion to Binary Decision Diagram. In Proceedings of the The First International Conference on Availability, Reliability and Security, ARES 2006, the International Dependability Conference—Bridging Theory and Practice, Vienna, Austria, 20–22 April 2006; pp. 449–457. [\[CrossRef\]](#)
20. Deng, Y.; Wang, H.; Guo, B. BDD algorithms based on modularization for fault tree analysis. *Prog. Nucl. Energy* **2015**, *85*, 192–199. [\[CrossRef\]](#)
21. Bitner, J.R.; Jain, J.; Abadir, M.S.; Abraham, J.A.; Fussell, D.S. Efficient Algorithmic Circuit Verification Using Indexed BDDs. In Proceedings of the Digest of Papers: FTCS/24, the Twenty-Fourth Annual International Symposium on Fault-Tolerant Computing, Austin, TX, USA, 15–17 June 1994; pp. 266–275. [\[CrossRef\]](#)
22. van Eijk, C.A.J. A BDD-based verification method for large synthesized circuits. *Integration* **1997**, *23*, 131–149. [\[CrossRef\]](#)
23. Scholl, C.; Möller, D.; Molitor, P.; Drechsler, R. BDD minimization using symmetries. *IEEE Trans. CAD Integr. Circuits Syst.* **1999**, *18*, 81–100. [\[CrossRef\]](#)
24. Drechsler, R.; Drechsler, N.; Günther, W. Fast exact minimization of BDD's. *IEEE Trans. CAD Integr. Circuits Syst.* **2000**, *19*, 384–389. [\[CrossRef\]](#)
25. Aloul, F.A.; Markov, I.L.; Sakallah, K.A. Efficient Gate and Input Ordering for Circuit-to-BDD Conversion. In Proceedings of the 11th IEEE/ACM International Workshop on Logic & Synthesis, IWLS 2002, New Orleans, LA, USA, 4–7 June 2002; pp. 137–142.
26. Fey, G.; Shi, J.; Drechsler, R. BDD Circuit Optimization for Path Delay Fault Testability. In Proceedings of the Euromicro Symposium on Digital Systems Design (DSD 2004), Architectures, Methods and Tools, Rennes, France, 31 August–3 September 2004; pp. 168–172. [\[CrossRef\]](#)
27. Ebdet, R.; Günther, W.; Drechsler, R. Combining ordered best-first search with branch and bound for exact BDD minimization. *IEEE Trans. CAD Integr. Circuits Syst.* **2005**, *24*, 1515–1529. [\[CrossRef\]](#)
28. Fey, G.; Drechsler, R. Minimizing the number of paths in BDDs: Theory and algorithm. *IEEE Trans. CAD Integr. Circuits Syst.* **2006**, *25*, 4–11. [\[CrossRef\]](#)
29. Ebdet, R.; Drechsler, R. Effect of improved lower bounds in dynamic BDD reordering. *IEEE Trans. CAD Integr. Circuits Syst.* **2006**, *25*, 902–909. [\[CrossRef\]](#)
30. Dinh, Q.; Chen, D.; Wong, M.D.F. BDD-based circuit restructuring for reducing dynamic power. In Proceedings of the 28th International Conference on Computer Design, ICCD 2010, Amsterdam, The Netherlands, 3–6 October 2010; pp. 548–554. [\[CrossRef\]](#)
31. Ubar, R.; Marenkov, M.; Mironov, D.; Viies, V. Modeling sequential circuits with shared structurally synthesized BDDs. In Proceedings of the 9th International Design and Test Symposium, IDT 2014, Algeria, Algeria, 16–18 December 2014; pp. 130–135. [\[CrossRef\]](#)
32. Rauchenecker, A.; Wille, R. An efficient physical design of fully-testable BDD-based circuits. In Proceedings of the 20th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2017, Dresden, Germany, 19–21 April 2017; pp. 6–11. [\[CrossRef\]](#)
33. Ubar, R.; Jürimägi, L.; Adekoya, A.O.; Jenihhin, M. True Path Tracing in Structurally Synthesized BDDs for Testability Analysis of Digital Circuits. In Proceedings of the 22nd Euromicro Conference on Digital System Design, DSD 2019, Kallithea, Greece, 28–30 August 2019; pp. 492–499. [\[CrossRef\]](#)
34. Matsuo, R.; Shiomi, J.; Ishihara, T.; Onodera, H.; Shinya, A.; Notomi, M. Methods for Reducing Power and Area of BDD-based Optical Logic Circuits. *IEICE Trans.* **2019**, *102-A*, 1751–1759. [\[CrossRef\]](#)
35. Narodytska, N.; Walsh, T. Constraint and Variable Ordering Heuristics for Compiling Configuration Problems. In Proceedings of the International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 149–154.
36. Trinidad, P.; Benavides, D.; Ruiz-Cortes, A.; Segura, S.; Jimenez, A. FAMA Framework. In Proceedings of the Software Product Line Conference, Limerick, Ireland, 8–12 September 2008; p. 359.
37. Mendonça, M. Efficient Reasoning Techniques for Large Scale Feature Models. Ph.D. Thesis, University of Waterloo, Waterloo, ON, Canada, 2009.
38. Czarnecki, K.; She, S.; Wasowski, A. Sample Spaces and Feature Models: There and Back Again. In Proceedings of the 12th International Software Product Line Conference, Limerick, Ireland, 8–12 September 2008; pp. 22–31. [\[CrossRef\]](#)

39. Classen, A.; Heymans, P.; Schobbens, P.; Legay, A. Symbolic model checking of software product lines. In Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu, HI, USA, 21–28 May 2011; pp. 321–330. [\[CrossRef\]](#)
40. Heradio, R.; Fernandez-Amoros, D.; Torre-Cubillo, L.; Garcia-Plaza, A.P. Improving the accuracy of COPLIMO to estimate the payoff of a software product line. *Expert Syst. Appl.* **2012**, *39*, 7919–7928. [\[CrossRef\]](#)
41. Heradio, R.; Fernandez-Amoros, D.; de la Torre, L.; Abad, I. Exemplar driven development of software product lines. *Expert Syst. Appl.* **2012**, *39*, 12885–12896. [\[CrossRef\]](#)
42. Beyer, D.; Stahlbauer, A. BDD-Based Software Model Checking with CPAchecker. In Proceedings of the Mathematical and Engineering Methods in Computer Science, 8th International Doctoral Workshop, MEMICS 2012, Znojmo, Czech Republic, 25–28 October 2012; Revised Selected Papers, pp. 1–11. [\[CrossRef\]](#)
43. Heradio, R.; Fernandez-Amoros, D.; Perez-Morago, H.; Adan, A. Speeding up Derivative Configuration from Product Platforms. *Entropy* **2014**, *16*, 3329–3356. [\[CrossRef\]](#)
44. Apel, S.; von Rhein, A.; Wendler, P.; Größlinger, A.; Beyer, D. Strategies for product-line verification: Case studies and experiments. In Proceedings of the 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, 18–26 May 2013; pp. 482–491. [\[CrossRef\]](#)
45. Heradio, R.; Perez-Morago, H.; Alferez, M.; Fernandez-Amoros, D.; Alferez, G.H. Augmenting measure sensitivity to detect essential, dispensable and highly incompatible features in mass customization. *Eur. J. Oper. Res.* **2016**, *248*, 1066–1077. [\[CrossRef\]](#)
46. Aloul, F.A.; Markov, I.L.; Sakallah, K.A. FORCE: A fast and easy-to-implement variable-ordering heuristic. In Proceedings of the 13th ACM Great Lakes Symposium on VLSI 2003, Washington, DC, USA, 28–29 April 2003; pp. 116–119. [\[CrossRef\]](#)
47. Aloul, F.A.; Markov, I.L.; Sakallah, K.A. MINCE: A Static Global Variable-Ordering Heuristic for SAT Search and BDD Manipulation. *J. UCS* **2004**, *10*, 1562–1596. [\[CrossRef\]](#)
48. Meinel, C.; Somenzi, F.; Theobald, T. Function Decomposition and Synthesis Using Linear Sifting. In Proceedings of the ASP-DAC '98, Asia and South Pacific Design Automation Conference 1998, Pacifico Yokohama, Yokohama, Japan, 10–13 February 1998; pp. 81–86. [\[CrossRef\]](#)
49. Rudell, R. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *The Best of ICCAD: 20 Years of Excellence in Computer-Aided Design*; Springer: Boston, MA, USA, 2003; pp. 51–63. [\[CrossRef\]](#)
50. Benavides, D.; Segura, S.; Ruiz-Cortes, A. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.* **2010**, *35*, 615–636. [\[CrossRef\]](#)
51. Fernandez-Amoros, D.; Heradio, R.; Cerrada, J.A.; Cerrada, C. A scalable approach to exact model and commonality counting for extended feature models. *IEEE Trans. Softw. Eng.* **2014**, *40*, 895–910. [\[CrossRef\]](#)
52. Heradio, R.; Perez-Morago, H.; Fernandez-Amoros, D.; Cabrerizo, F.J.; Herrera-Viedma, E. A bibliometric analysis of 20 years of research on software product lines. *Inf. Softw. Technol.* **2016**, *72*, 1–15. [\[CrossRef\]](#)
53. Berger, T.; She, S.; Lotufo, R.; Wasowski, A.; Czarnecki, K. A Study of Variability Models and Languages in the Systems Software Domain. *IEEE Trans. Softw. Eng.* **2013**, *39*, 1611–1640. [\[CrossRef\]](#)
54. Biere, A.; Le Berre, D.; Lonca, E.; Manthey, N. Detecting Cardinality Constraints in CNF. In *Theory and Applications of Satisfiability Testing—SAT 2014*; Sinz, C., Egly, U., Eds.; Springer International Publishing: Cham, Switzerland, 2014; pp. 285–301.
55. She, S.; Berger, T. *Formal Semantics of the Kconfig Language*; Technical Report; University of Waterloo: Waterloo, ON, Canada, 2010.
56. She, S. Feature Model Synthesis. Ph.D. Thesis, University of Waterloo: Waterloo, ON, Canada, 2013.
57. Berger, T.; She, S.; Lotufo, R.; Wasowski, A.; Czarnecki, K. Variability modeling in the real: A perspective from the operating systems domain. In Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Antwerp, Belgium, 20–24 September 2010; ACM, IEEE: Lawrence, KS, USA, 2010; pp. 73–82.
58. Zengler, C.; Küchlin, W. Encoding the Linux kernel configuration in propositional logic. In Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010) Workshop on Configuration, Lisbon, Portugal, 16–20 August 2010; IOS Press: Amsterdam, The Netherlands, 2010; Volume 2010, pp. 51–56.
59. Walch, M.; Walter, R.; Küchlin, W. Formal Analysis of the Linux Kernel Configuration with SAT Solving. In Proceedings of the 17th International Configuration Workshop, Vienna, Austria, 10–11 September 2015; University of Helsinki: Helsinki, Finland, 2015; pp. 131–137.

60. Fernandez-Amoros, D.; Heradio, R.; Cerrada, C.; Herrera-Viedma, E.; Cobo Manuel, J. Towards Taming Variability Models in the Wild. In *New Trends in Intelligent Software Methodologies, Tools and Techniques, Proceedings of the 16th International Conference SoMeT_17, Granada, Spain, 26–28 September 2017*; IOS Press: Amsterdam, The Netherlands, 2017; Volume 297, p. 454.
61. Kästner, C. Differential Testing for Variational Analyses: Experience from Developing KConfigReader. *arXiv* **2017**, arXiv:1706.09357.
62. Brace, K.S.; Rudell, R.L.; Bryant, R.E. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference, Orlando, FL, USA, 24–28 June 1990*; pp. 40–45. [[CrossRef](#)]
63. Krieter, S.; Thüm, T.; Schulze, S.; Schröter, R.; Saake, G. Propagating Configuration Decisions with Modal Implication Graphs. In *Proceedings of the 40th International Conference on Software Engineering, ICSE '18, Gothenburg, Sweden, 17 May–3 June 2018*; ACM: New York, NY, USA, 2018; pp. 898–909. [[CrossRef](#)]
64. Nohrer, A.; Egyed, A. Optimizing user guidance during decision-making. In *Proceedings of the IEEE 15th International Software Product Line Conference, Munich, Germany, 22–26 August 2011*; pp. 25–34.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).