GR Circuit - Functional Macro Blocks Circuit Recovering Tool

Leandro M. Silva · Fabricio V. Andrade · Luiz F. M. Vieira

Received: date / Accepted: date

Abstract Considering that the more information you can gather about a particular circuit, you can address problems more accurately in the Electronic Design Automation (EDA) field, therefore, many tools focus on obtaining the maximum amount of information about the input to which it is provided in order to determine which are the best algorithms to each instance. Some of these tools are the Boolean Satisfiability (SAT) problem solvers; which, for the most part, receive formulas described in Conjunctive Normal Form (CNF) as input. The circuits encoding process to the CNF format, unfortunately, destroy much of the information that could have been used to optimize SAT solvers, as part of this informations must be recovered to avoid applying generic algorithms in the solution of SAT problems. One of the difficult aspects of retrieving this information corresponds to the matching of clauses to its respective logic gates, as well as which sets of logic gates correlate to a functional block. The present work makes use of subgraph isomorphism algorithms to recover circuits encoded in CNF-DIMACS maximizing the number of clauses handled, both at the level of logic gates as well as more complex structural blocks, which allow their identification at higher levels of abstraction. Our tool was able to successfully recover all circuits generated as tests, in addition to recover 100% of the circuits from ISCAS 85 and ITC 99 benchmarks.

Keywords circuit recovering · circuit’ structural information · subgraph isomorphism · SAT

Acknowledgements The authors would like to thank the research agencies Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) for their financial support and NVIDIA for equipment donation.

1 Introduction

The constant advances in Boolean Satisfaction (SAT) algorithms have allowed SAT solvers to become routinely applied in formal verification and circuit testing; especially in Equivalence Checking, Bounded Model Checking (BMC), Automatic Test Pattern Generation (ATPG) and microprocessor verification. Generic SAT resolvers have brought gradual results, some, like GRASP [21], Chaff [17] and BerkMin BerkMin [11], have improved performance by several orders of magnitude on various benchmarks derived from applications. These good performances in NP-complete problems generated a series of new applications, not only to the Electronic Design Automation (EDA), but also to the Artificial Intelligence (AI) and optimization field, mainly in planning and scheduling. This good performance in NP-complete problems generated a series of new applications, not only in Electronic Design Automation (EDA) but also in Artificial Intelligence (AI) and optimization, mainly in planning and scheduling. In scenarios where circuit based applications are found, generic SAT resolvers have been extended to be applied in Conjunc-
tive Normal Form (CNF) instance inputs derived from the circuits [16]. The execution time was also improved by an order of magnitude for some classes of benchmarks, but with the caveat that the circuit structure must be known before the resolution. Among the main techniques that use the circuit structure are 1) the parallel simulation of a small number of random inputs; 2) the detection of correlation between the signals using hash and; 3) a guide for SAT resolvers to refute the equivalence of related signals and thus help you generate more concise and efficient conflict clauses. When using the information of the circuit structure, the authors assume that the information present in it was lost during the conversion to a CNF-SAT [16] instance.

Some CNF formulas can contain a large number of clauses derived from circuits although, they did not originate from circuits. An example of this type of formula is property verification, where the circuit part describes the hardware and the non-circuit part represents more general properties, such as “at most k of n signals can be high”. Besides, several circuits can appear in the form of mathematical properties, without the knowledge of the designer who performed the coding.

Whereas many SAT instances are not derived from circuits, other factors, such as the involvement of un-oriented gates (i.e. XOR), make it harder to recover the circuit present in the formula. Even in these cases, Roy et al. [19] have shown that parts of the CNF formulas compatible with the circuit structure can be found, and that the recognition of circuit parts facilitates the SAT resolvers in their operations, regardless of whether the reconstructed circuits are unique.

In view of SAT solvers efficiency solving SAT problems, they have become a standard tool in many applications. Despite the fact that some problems belong to the NP class, the solvers have proved their competence solving practical problems on acceptable periods.

1.1 Motivation

Most SAT resolvers expect the input to be in CNF, more precisely in CNF-DIMACS [12]. When dealing with electronic circuits, it is possible to encode the CNF circuit in linear time using the Tseitin [25] transformation, which is the most used coding. Although simple and efficient, this transformation flattens the circuit, destroying its topology, causing the logic gates and their connections to be lost and an equivalent circuit composed of a large AND with ORs and inverters connected to it is used instead. The lost information of the circuit structure can be useful for SAT resolvers that benefit from this information.

The structural recovery work of the CNF-encoded circuit may seem unnecessary since the SAT resolver could use as information the original circuit from which the CNF was produced. In an ideal scenario, this is the most correct and efficient process. However, this circuit may not always be present. Two very different scenarios where the extraction of information is useful due to the absence of the original circuit: 1) the CNF is not generated from a circuit, such as planning and scheduling and routing in FPGAs; 2) the CNF is generated from a circuit but it is not available, as is the case with most of the benchmarks used in SAT solver competitions.

Finally, to the best of our knowledge, any type of extra information related to the circuit structure present in the formula, regardless of whether the origin of the circuit is an electronic circuit, boosted the performance of the resolvers which have used them.

1.2 Our Contributions

Our work presents a method of structural recovery of functional macro blocks based on graph isomorphism. Among the main contributions there are three we would like to highlight: 1) the possibility of operating structural recovery and identification of both, the input and output signals of the functional block without needing to detect the logic gates it composes or its directions; 2) a tool that enables its users to determine which blocks of functionalities they want to detect without any changes to its source code; and 3) a highly accurate tool that was tested in extensive simulations in different sets of circuits and was able to recover 100% of the circuits from ISCAS85 and ITC 99 benchmarks.

1.3 Roadmap

The following Section presents the related work. Section 3 describes the circuit recovery. Section 4 details our experiments and results. Finally, in Section 5 we present our conclusions.

2 Related Work

During Section 1, we saw that, in general, loss of information is inevitable when coding a circuit in CNF. Perhaps due to this consideration, there were no meaningful efforts in extracting the circuit structure from CNF.

The earliest works on the structural recovery line, that we are aware of, were the extraction of equivalences between literals [14] and simple AND and OR [18] gates.
Roy et al. were the first to explicitly extract logic gates from CNF [19]. They introduced the concept of CNF signature, which in a whisper is the CNF encoding of a logic gate. This signature is transformed into a graph and isomorphism algorithms in subgraphs are used to find correspondences of these signatures in the complete graph. Few implementation details are provided and clearly the focus of the work is to find the basic logic gates (AND, OR, NAND, NOR, XOR, XNOR and inverter), but impose strong restrictions on the occurrences of XOR / XNOR gates in the extraction process.

Later work by Zhaohui Fu and Sharad Malik [10] is based on a library of gates that describes the patterns to be extracted. The use of this library guarantees a more flexible approach, but less efficient than specific pattern matching for door types. Another contribution of this work is that, according to the authors, they not only extract logic gates but also guarantee to extract the largest acyclic circuit possible through the use of SAT resolvers.

More recently, Harald Seltner [20] defended his master’s thesis developed with a strong theoretical foundation in the work of Zhaohui Fu. As a main result, a tool called cnf2aig was developed that reconstructs circuits from CNF and outputs And-Inverter Graphs (AIG) [4] as output. The author also guarantees that the reconstructed circuit is as much close as possible, with respect to the gates, that their algorithms can detect.

The structural information is very useful in other types of problems that are not related to SAT. An example of this is the work of Chakraborty [5] whose identification of certain multiplier structures and the addition of special assertions to the input formula allowed the solution of several formulas quickly where they would usually have timeout, despite Satisfiability Modulo Theories (SMT) solvers are known to be inefficient in formulas with bit vectors in multipliers.

The recovery of lost information about the circuits is so important that several studies try to obscure the information so that it is not used in reverse engineering. The works of Liu et. al [15] and Yu et. al [26] present the use of SAT-solvers to perform reverse engineering on logic circuits that had part of their circuit camouflaged. They assume that only part of the circuit is camouflaged and each camouflage generates a few different and equivalent logical functions. The work of Keshavarz et. al [13] shows that structural recovery techniques using SAT in obfuscated circuits generally fail to recover correct gate-level schematic even if able to control inputs and probe all combinational nodes of the circuit. Therefore, they show a more aggressive way that extend the attacks using SAT and with that they manage to recover the circuit. The work of Tan et. al [24] goes in the opposite direction to the other works. They analyzes the application of random errors in the circuits as a mechanism of reverse engineering.

As demonstrated, structural recovery brings important information for the vast majority of applications, especially when compared when applying generic algorithms where the circuit is considered a generic formula like any other. For this reason, several studies try to hide this information, mainly due to Intellectual Property (IP), while others try to recover it to apply their techniques (i.e. Circuit Equivalence Checking).

Regarding obfuscated circuits, our work can deal with some of them as long as their equivalent logical function is present in our library. Since this is not part of the scope of this paper, we do not evaluate these types of circuits.

In our first work in the area of structural recovery [22], we used neural networks to identify the type of circuit present at the CNF entrance. In this work, the goal was only to identify the type of circuit as a whole, without recovering the connections. As a continuation of this work, we made a more elaborate identification using a pre-processing to extract the characteristics used by the neural network in the identification [23]. Like the previous work, the objective was to identify the entire circuit. Part of how these characteristics are extracted is present in the current work.

Our work offers improvements in relation to the above mentioned developments because, in addition to detecting the entire circuit, or logic gates, as the previous works also did, it also allows recognizing functional macro blocks. Furthermore, when detecting the macro block, it is often possible to identify the direction of the input and output signals without needing to do it for all the internal gates in the analysed block. To the best of our knowledge, neither were done in any of the previous works.

## 3 Circuit Recovering

The encoding of combinational circuits in CNF-DIMACS is very direct and in linear time [25]. As any combinational circuit can be described as a sum-of-products (Disjunctive Normal Form - DNF) or product-of-sums (Conjunctive Normal Form - CNF), it is always possible to write it using only the three basic Boolean operators: OR (\(\lor\)), AND (\(\land\)) and NOT (\(\neg\)). A Boolean variable can only assume the values TRUE or FALSE. In a CNF formula, a clause is a disjunction of several variables, which may or may not be negated. An expression in CNF is, therefore, a set of one or more clauses. As seen in the 1 section, the most common input for-
Although a logic gate can have many CNF formulas, a given formula identifies no more than a logic gate. For this reason, we will call a formula by CNF-Signature. Once a CNF-Signature has been identified, it is guaranteed that a logic port has been found.

To find out all the occurrences of a given logic gate in a circuit instance, we can reduce our problem to the subgraph isomorphism problem, as we will see below. The subgraph isomorphism problem is known as being NP-complete for graphs in general, but there are several practical algorithms with good performance [8][6].

Every CNF formula can be expressed using an undirected graph without any loss of information. We will show how this is possible by construction, summarizing the process described by Roy et. al [19].

- Let $C$ be the set of all clauses of the formula, such that a clause $c_i \in C$;
- Let $X$ be the set of all variables in the formula, such that a variable $x_i \in X$;
- Let $L$ be the set of literals such that $l_+^i$ is the positive literal of the variable $x_i$ and $l_-^i$ is the negative literal of the same variable $x_i$;
- $\forall c_i; c_i \in C$, create a vertex labeled $vc_i$ for $c_i$;
- $\forall l_+^i, l_-^i \in L$, create a vertex labeled $v_{+i}$ and a vertex labeled $v_{-i}$, referring to $l_+^i$ and $l_-^i$, respectively, and create an edge $(v_{+i}, v_{-i})$;
- $\forall l_+^i \in c_j$, create an edge $(vc_j, v_{+i})$;
- $\forall l_-^i \in c_j$, create an edge $(vc_j, v_{-i})$;

At the end of this process, you will have an undirected labeled graph with three possible labels: clause vertex $(vc_i)$, positive literal vertex $(v_{+i})$ and negative literal vertex $(v_{-i})$. It is easy to see that all the information present in the CNF formula is still present in the generated graph and can easily be retrieved from it.

3.1 CNF-Signature

We saw in the previous section how to transform a CNF formula into an undirected labeled graph. We have also seen that every logical port has a characteristic function that, once identified, unequivocally represents the logical port associated with the function. We call CNF-Signature the formula that represents the logic gate and we will use the graph created from the characteristic function to identify its occurrence within the circuit instance. Although a logic gate has many CNF-Signatures, all graphs formed from them are isomorphs with each other, which means that when identifying a subgraph corresponding to a signature in a graph representing the circuit instance, a characteristic function...
is identified. This characteristic is very important because it allows the identification of logic gates using subgraphs isomorphism algorithms. Once the subgraph is identified, it represents only a characteristic function, and consequently, a logic gate.

Figure 2 shows examples of CNF-Signatures and their respective graphs for some basic logic gates.

3.2 Logic Gates Matching using Subgraph Isomorphism

In the previous section, we saw that when creating a graph representing the circuit instance and graphs from the CNF-Signatures of the logic gates, we can use isomorphism algorithms in subgraphs to locate the occurrence of these signatures in the circuit instance. The implementation of this best-known algorithm with excellent performance is present in VFLib [8]. The big problem with using the algorithm directly is the number of existing matches for the same graph since there are \( \binom{n}{k} \) for a gate with \( n \) inputs. In addition to the previous problem, easily soluble by marking the vertices already used, previous works that mention the use of matching using isomorphism do not show how to solve the matching problem of a subgraph of a logic gate with fewer entries in a graph of a logic gate with more entries. Roy et. al [19] cites the matching through isomorphism in subgraphs but uses a different algorithm for detecting the gates. To illustrate the problem, consider the graphs of three AND logic gates with 4, 3 and 2 inputs in Figure 3.

Note that, with even the definition of labeled graph differentiating the vertices of clauses (red vertices), positive literals (blue vertices) and negative (orange vertices), the 3-AND and 2-AND graphs are 4-AND subgraphs, as well as 2-AND is a subgraph of 3-AND. We can see that if there is a 4-AND gate, all input combinations are matched, even though only the 4-AND gate is the correct match. To solve the problem of matching logic gates with fewer inputs, we add extra information to each clause vertex that corresponds to the number of literals to which the clause is linked. As, by definition of a labeled graph, there is a match only if the vertices have the same properties, a clause vertex with 5 literals does not match another one with 4 literals, and so on. The addition of the number of literals linked to the clauses, in order to solve the problem of smaller gates being identified where larger gates are the correct correspondence, brings another contribution of our work: the possibility of identifying the NOT and BUFFER ports through isomorphism in subgraphs, not possible through previous works.

3.3 Complex or Parametrizable Blocks Matching

In circuits, the connection between logic gates is made using wires, connecting the output of a logic gate to an input of another logic gate. In CNF, each logical port is encoded in a set of clauses, and a clause belongs exclusively to a single logic gate. For this reason, the connection between two logic gates can only be made by a variable that represents the output of one gate and the input of another gate at the same time.

The alluded observation is important because two subgraphs can only be connected through vertexes of literals, never vertices of clauses, as two ports can only be connected by wires, which are represented by variables. This sort of construction, therefore, empowers us to build signatures of any size that will represent any functional blocks. From now on, functional blocks built from logic gates, or other blocks, will be named complex blocks.

Some complex blocks can also be defined by a certain degree of recursion or by a logic that clearly depends on a parameter (e.g. the number of bits). A circuit that has this type of construction is the Ripple Carry Adder (RCA) because the connection of the logic blocks is always made by connecting the signal \( \text{cout} \) of the previous full-adder and the \( \text{cin} \) of the current full-adder. Some blocks are not as straightforward as the
RCA but their recursive construction is easily identifiable, as in the case of the Carry Lookahed Generator (CLG) present in the Clarry Lookahead Adders. These types of blocks will be named Parametrizable Blocks. Both types of blocks are fundamental for this work since several CNF-Signatures used were built from complex or parameterizable blocks.

3.4 Gate Direction Identification

As we saw in the 3.2 section, once a subgraph is identified, it represents only a logical gate. Although the gate is identified, we do not know which of the variables represent the inputs and which the output. For some logic gates, this information is obtained directly from the subgraph. For example, consider again the AND gate graph shown in Figure 2. The only clause with more than two variables (Clause 1) is the clause that contains the information of which variable corresponds to the output, according to the definition of the signature, because the only variable that appears not negated is the output variable, in this case, represented by literal 3 (variable c).

Similarly, looking at the graph of the OR gate (Figure 2) we can see that Clause 1 is linked to the gate variables and only the literal -3 (variable c) is in the negated form. Thus, detecting the direction of the gates for AND and OR is trivial. However, this cannot be done for other gates, for instance, NAND and NOR. Although this direction cannot be obtained so directly, a simple algorithm allows this detection, analyzing how the variables appear in the other clauses of the signature.

The XOR and XNOR logic gates are not oriented. This means that, analyzing exclusively your CNF-Signatures or graph, it is not possible to identify which variables correspond to the inputs and which output. The same happens with the inverter (NOT) and the buffer. For these logic gates, it is necessary that they are connected to other logic gates. Their direction is possible to be identified, thus allowing their identification.

Consider a slightly more complex example, the circuit of a full-adder illustrated in Figure 4.

Through CNF-signatures, the logical ports are correctly identified. The next step is the identification of directions. As the two ANDs and the OR are gates whose direction is directly determinable, we conclude:
For OR, 7 and 8 are inputs and 5 is output.

For the first AND, 3 and 6 are inputs and 7 is output.

For the second AND, 1 and 2 are inputs and 8 is output.

Based on the directions previously determined, it is not yet possible to complete the direction of the two remaining XOR gates, because:

Even though 3 and 6 are inputs from the first AND, 4 and 6 could be primary inputs, 3 output from XOR;

1 and 6 primary inputs and 2 XOR outputs, or 2 and 6 primary inputs and 1 XOR output.

An important factor is that a variable can only be associated with the output of a single port. Based on this restriction, once identified as an output from a port, all occurrences of that variable in other ports indicate that it corresponds to an input.

Another factor is that in order to determine the direction of unoriented gates, they must be interspersed with a gate whose direction can be determined.

However, the constructions of the CNF-Signatures are made in such a way that they are located in the circuits that we want to make the restoration. As the signatures are constructed, our work added some extra information to the signatures so that we know, when the vertex is identified, if it corresponds to an input, output or internal vertex.

Consider as an example the CNF-Signature graph of the NAND logic port, in Figure 2. Only by performing the match, it is not possible to identify which signals are entered and which is output, since the key clause that contains all literals, they are all negated. It is only possible to identify the direction by analyzing the occurrence of literals in the other clauses.

Now suppose we add extra markup to the vertices that correspond to input and output signals when creating a CNF-Signature. When the match is performed, the vertices that match the marked vertices will correspond to the input and output variables. This extra information makes it possible for the direction of the doors to be identified only with subgraph isomorphism for the AND and OR gates as done by Roy et. al, and also for the NAND and NOR ports.

The addition of the extra information of which vertices correspond to the inputs and outputs adds little contribution when used only with logic gates, but adds enormous gain when applied to complex and parameterizable blocks. Let the full-adder in Figure 4. We argue that only the logic gates are identified, it is not possible to identify the correct direction of the signals in the circuit. Now imagine that we have a CNF-Signature that the vertices referring to variables 1, 2 and 3 are marked as input and 4 and 5 as output. When a match takes place in a circuit, the variables that match the marked vertices will have their direction identified, without the need to identify the directions of the intermediate signals, since only the inputs and outputs of the functional block are of interest. Figure 5 shows the CNF-Signature graph of a Complete Adder with the marked vertices.

The identification of complex and/or parameterizable blocks associated with the direction of the signals of these blocks are among the main contributions of the present work. Within the limits of our knowledge, no previous work has shown how to build signatures or identify functional blocks, nor how to identify the direction of signals.

3.5 Conflict Resolution

Although a matching corresponds only to a characteristic function, it does not mean that this marriage, even if valid, corresponds exactly to a function present in the circuit. Figure 6 illustrates an excerpt from the c1355 circuit present in the ISCAS 85 Benchmark:

These clauses are part of a real Single-Error-Correcting Circuit present in ISCAS 85 [9] and are Tseitin transformations of logic gates $266 = \text{NAND}(1, 8)$ and $363 = \text{NAND}(8,266)$. In spite of this, if we take into account only these six clauses, the combination of clauses 25, 123 and 125 allows the matching with logic port $8 = \text{NAND}(266,363)$, which is not an existing port in the original circuit. Despite the fact that a valid matching does not correspond to a real matching, it still prevents correct matching from happening, as there is no sharing of clauses between different logic gates. For this reason, if the aforementioned matching is identified earlier, it prevents clause 25 from being used on one gate and clauses 123 and 125 on another.

To solve this problem, we created a conflict graph. When there is a conflict between matchings, an edge
connecting the matching is created. The existence of this edge indicates that some clause is shared between the two matchings. The solution for the reconstruction of the circuit without conflicts is the solution to the problem of the Maximum Independent Set (MIS), as indicated by Roy et al. [19]. The problem with MIS is NP-Hard, but there are good and efficient heuristics. In addition, our experiments show that, although some instances of circuits are very large, the number of conflicts is very small, allowing solutions to be found quickly.

Our work applies a variation of MIS known as Maximum Weight Independent Set (MWIs), whose vertices have a weight associated and output consists of vertices list that makes up an independent set with a maximum total weight. MIS is a specific case of MWIS where all vertices have equal weight. Our variation consists of associating with the vertex of the conflict or the number of clauses it restores. This variation is important because there is no work by Roy, the MIS found determines the largest recoverable circuit in number of logic gates, while our work finds the largest recoverable circuit in terms of the number of clauses.

The change in the use of MIS by MWIS ensures that it is maximum in the number of clauses used in the restoration of the circuit, ensuring that it is the maximum possible among all others. In addition, with MWIS it is possible to use libraries of CNF-Signatures that have signatures where one CNF-Signature is a subset of others, i.e. XOR and Full-Adder in the same library.

To illustrate the difference, be the circuit formed by a Full-Adder (Figure 4). This circuit has two 2-XOR, two 2-AND and one 2-OR. It is evident that the Full-Adder subscription conflicts with all the others. Figure 7 shows the graph of conflicts between marriages. If MIS were used, the Full-Adder match would never be selected because the choice of logic gates would return 5 vertices for the independent set, while the choice of Full-Adder would return only a single vertex. When using MWIS, the set with matches 1 to 5 has the same result as the set with only match 6.

Another gain that the use of MWIS has brought in relation to previous works is the possibility of using CNF-Signatures that are not direct Tseitin translations or that are built from gates that may not be part of the library. To illustrate, let it be the conflict graph presented in Figure 7. Suppose we have 2-AND, 2-XOR, Full-Adder signatures in our library but we do not have 2-OR signatures. When applying our methodology, the same graph would be formed with the exception of Match 5 (colored vertex). The result of the MWIS would return only the apex of Match 6, as it has a weight of 17, while the sum of the others, despite 4 vertices, has a total weight of 14.

3.6 Insoluble Situations

In some situations, only the information present in the circuit’s CNF files does not allow you to discover with complete certainty what the original circuit was. An example of an unsolvable situation is present in the bench-
mark ISCAS 85 c432 (Channel Interrupt Controller) circuit [9]. Let the following clauses, extracted from the said circuit, correspond to ports 118 = NOT (1) and 154 = NAND (118, 4), shown in Figure 8:

Although there is no conflict between matchings, the NOT is a gate whose direction cannot be identified without having its neighborhood analyzed, as we saw earlier. Signal 118 is correctly identified as input to the NAND gate, but it is not possible to define that, even though it is input from the NAND, it corresponds to an output, since signal 118 could be a primary input to the circuit, therefore input to NAND and NOT, and signal 1 would be output from the NOT port. To find the direction of the NOT gate, it needs to be in a context where one of its signals is guaranteed to be an output signal from another gate. In the example of circuit c432, signal 1 is only connected to gate 242 = NAND (1, 213), not even allowing to know if 1 is NOT output or if it is a primary input connected to NOT and NAND. Figure 8c shows another possible circuit that has exactly the same clauses as the original circuit.

It is important to emphasize this situation because in the transformation of the circuit to CNF using Tseitin, it is not only the logic gates and their connections that are lost. The information of which are the primary inputs and outputs, or their quantities, prevents, in some cases, the possibility of asserting the real circuit of origin and for the case of c432, both solutions are equally valid.

3.7 Recovering process

Figure 9 show us the synthesis of the recovery process using GRCircuit tool. A unknown circuit encoded in CNF-DIMACS format is provided as input to the tool, which uses a pre-generated library with CNF-Signatures and process the input in order to recover it.

The output consists of two files: 1) the maximum recovered circuit according to the CNF-Signature library, in number of clauses used and 2) the set of unused clauses, that is, clauses that did not match any signature present in the library.

4 Experiments and Results

In our experiments, real circuit designs used by the EDA industry were used. These circuits were generated by the Bencgen tool [2] whose objective is to supply circuits of industrial standards to be used in tool benchmarks. According to the authors of the tool, it is possible to generate circuits of different types from five different classes and different numbers of bits, totaling more than one and a half million circuits. Despite the number of circuits available, they can be generated in the formats BENCH [9], BLIF [3], EQN and in the newer version, Verilog [1] and CNF-DIMACS [12]. The Verilog and CNF-DIMACS formats are particularly interesting because in the former we have the information necessary to build our CNF-Signature library, while the latter are sources of circuits for our primary tests. Table 1 shows a summary of circuits, their classes and the maximum number of bits, generated and used to create the base of our library and tests. These 8,852 circuits correspond to several varieties of logic and arithmetic circuits present in real industrial circuits and in the vast majority of Arithmetic Logic Units (ALUs).

For the construction of our library, we generated circuits in Verilog format in 21 classes, limiting the circuit size to 1) 2-bit powers or 2) maximum time of 60 seconds, whichever came first. This approach allowed us to generate a total of 8,755 different circuits. With these circuits, we extracted all the modules present in these circuits, totaling a collection of 1,190 different modules. The collected modules were converted to CNF-DIMACS and a further 510 files were created with the CNF encoding of the basic logic gates up to 128 bits (127 ANDs, 127 ORs, 127 NANDs, 127 NORs, 1 XOR and 1 XNOR), totaling 1,700 signatures. As shown in Section 3.2, the input and output variables were marked so that signatures can be used properly in our work.

To perform our tests, we also used the tool to generate the circuits, following the same criteria but using CNF-DIMACS as the output format. These circuits
form the basis of our primary tests and have not undergone any changes and/or markings, being used exactly as made available.

Table 2 shows the absolute numbers of clauses identified through the application of the basic logic gate identification methods and the methods for identifying their direction.

The application of our method using our library of basic logic gates was able to identify 1,014,927,458 clauses, corresponding to 89.6744% of all 1,131,791,747 clauses. The values were the same for the directions noting that, for all the detected logic gates, it was possible to identify the direction of the signals, that is, the exceptions that prevent the directions from being identified (section 3.1) were not found on the tested circuits.

Table 3 shows that using the complete CNF-Signatures library, not just the basic logic gate signatures, totaling 1,700, we were able to identify 82.9831% of all clauses as present in some module and 9.4171% of clauses not present in modules, that is, basic logic gates. The result showed that 100% of all identified modules and logic gates had their input and output signals identified.

If we compare the results of Table 2 with Table 3 we can verify that the circuits of types C13 and C21 had improvement in the identification results, showing that the basic logic gates that could not be identified separately had their clauses identified within the modules. The increase in the percentage of recognition from 89.6744% to 92.4002% shows this improvement.

In order to identify the reason for not identifying all clauses in circuits C43, C44, C45 and C48, we made an analysis of the logic gates identified in all circuits and identified that the probable problem was in the AND and OR logic gates.

Table 4 shows the number of logical gates identified by type up to the limit of 128 bits, our initial library. Columns with an upper limit of 128 bits show how many more gates were identified when increasing the limit.

We were able to identify that for our test suite, there are logical AND gates of up to 8192 bits. For the OR logic gate with a limit of 8192, not all clauses were identified, but we could see a trend of growth in power of 2, evidenced by the C45 circuit. A manual analysis showed that there are ORs from 214 to 220, so that all clauses of all circuits were identified.

Despite the validity, logically, of logic gates with many inputs, they are not present in real circuits. For this reason, we apply a transformation of logical AND and OR gates greater than 128 bits, transforming them into ANDs and ORs limited to 128 bits to perform our tests.

Table 5 presents the synthesis of the number of detected logical gate occurrences and the number of modules per circuit type. Also shown is the number of different CNF-Signatures used for each type of circuit. Not all of the signatures in our library were used, as not all logic gates appeared in the entire bit range.

The analysis of the results presented shows that the library built for the tests presented a wide range. Some modules had a very high number of occurrences and percentage of clauses used, such as Full-Adder, Half-Adder in several circuits; and blocks A and S in the multipliers. In addition to the contribution of the created CNF-Signatures library, we were able to demonstrate that the proposed method of detecting complex functional blocks and the direction of the input and output signals proved to be very effective, with superior detection compared only to the use of basic logic gates.

The detection of the complete functional block eliminates the need to build connections between the logic gates internal to the block, increasing the efficiency of the circuit reconstruction, evidenced by the reconstruction of 100% of the circuits tested.

In order to validate our methodology and evaluate the scope of our CNF-Signatures library, we performed tests with benchmark circuits present in ISCAS85 and also in ITC 99 [7]. Table 6 shows the circuits used in our experiments. The variables and clauses columns are the result of the direct transformation of the logic gates into CNF using the Tseitin transformation presented in the 3 section. The flip-flops present in the circuits were not encoded, since it generates a combinational loop. From the point of view of the CNF clauses, this is not a problem, since the input of the FF is considered a pri-

---

**Table 1: Summary of circuits and the respective number of different CNF-Signatures**

| Circuit | Circuit Name | Class | N. Circuits |
|---------|--------------|-------|-------------|
| C11     | Ripple Carry Adder | Adder | 1,022 |
| C12     | Ripple Carry Adder/Subtractor | Adder | 1,022 |
| C13     | Carry-Lookahead Adder | Adder | 198 |
| C14     | Carry-Skip Adder | Adder | 1,022 |
| C15     | Carry-Select Adder | Adder | 1,022 |
| C16     | Carry-Save Adder | Adder | 1,022 |
| C17     | Carry-Lookahead Block Adder | Adder | 1,022 |
| C21     | Dadda Tree Multiplier | Multiplier | 126 |
| C24     | Array Multiplier | Multiplier | 254 |
| C25     | Carry-Save Multiplier | Multiplier | 254 |
| C31     | Non-Restoring Cellular Array Divider | Divider | 254 |
| C32     | Restoring Cellular Array Divider | Divider | 254 |
| C33     | Carry-Lookahead Array Divider | Divider | 162 |
| C42     | Decoder | Basic | 18 |
| C43     | Encoder | Basic | 18 |
| C44     | Multiplexer | Basic | 18 |
| C45     | Demultiplexer | Basic | 18 |
| C46     | Parity Circuit | Basic | 510 |
| C47     | Magnitude Comparator | Basic | 510 |
| C50     | Arithmetic Logic Unit | ALU | 126 |
| Total   |              |       | 8,852 |
We applied our methodology associated with our CNF-Signatures library built during our primary tests on the ISCAS 85 and ITC 99 benchmark circuits. Table 7 shows the number of each logic gate identified in each circuit. The gate XNOR is not present in any circuit, so its column was suppressed, as well gate direction for auxiliary algorithms depending on the gate type.

As can be seen from the results presented, our methodology allows the identification of basic logic gates using only subgraph isomorphism algorithms, with no need for auxiliary algorithms depending on the gate type. In addition, our methodology also allows signatures of more complex modules to be used, allowing the identification of functional blocks instead of just logic gates. This result, besides allowing a higher level of identification, yet allows the direction of the input and output signals to be more easily identified depending on the construction of the block. The finally, the form of selection of the clauses that are used in the circuit recovering process, like in C499 circuit. Although easy to identify, determining the directions of the signals is a real challenge. A lot of NOTs and Buffers are also a problem because their clauses generates conflict with others gates’ clauses, since its small clauses are easily considered as part of other logic gates.
maximizes the number of clauses used and allows complex or parameterizable functional blocks to be identified without having the CNF-Signatures of internal modules or gates.

5 Conclusion

In this paper, we aimed at presenting a form of structural recovering of circuits encoded in CNF-DIMACS using exclusively graph algorithms. Our work consists of basically two parts. The first consists in the construction of a library of CNF-Signatures containing a large collection of signatures of logic gates of various sizes, very common functional blocks (i.e. Full-Adder, Half-Adder and etc.), in addition to complex and parameterizable functional blocks. This library represents the starting point in identifying the circuits. The second and more complex corresponds to the implementation of a careful methodology for identifying circuits using only graph algorithms, culminating in the GRCircuit.
tool. The markings on the signatures and the labeled graphs with the appropriate information allowed the identification and direction of the signals that correspond to the logic gates and/or the functional blocks. Conflict cases are resolved using Maximum Weight Independent Set (MWIS), ensuring maximum use of the clauses.

Our results display the application of subgraph isomorphism in graphs that have been properly modeled corresponds to a powerful circuit identification tool, allowing the identification not only of logic gates, but also of substantially more complex functional blocks.

In conclusion, we intend to continue working to improve our CNF-Signature library as well as the algorithms performance and, in future research, pre-process the input in order to select a good subset of the library to be used as the first approach.

6 Declarations

6.1 Funding
Not applicable.

6.2 Conflicts of interest
The authors declare that they have no conflict of interest.

6.3 Availability of data and material
The circuits used in our experiments can be found at https://github.com/elmaia/cnf/tree/master/circuits

6.4 Code availability
Not applicable.

6.5 Authors’ contributions
Not applicable.

6.6 Ethics approval
Not applicable.

6.7 Consent to participate
Not applicable.

References

1. Standard for verilog hardware description language (2006). DOI 10.1109/IEEESTD.2006.99495
2. Andrade, F.V., Silva, L.M., Fernandes, A.O.: Benchgen: a digital circuit generation tool for benchmarks. In: SBCCI ’08: Proceedings of the 21st annual symposium on Integrated circuits and system design, pp. 164–169. ACM, New York, NY, USA (2008). DOI http://doi.acm.org/10.1145/1404371.1404418
3. Berkeley: Berkeley logic interchange format (blif) – http://www.cs.uic.edu/~jlillis/courses/cs594/spring05/blif.pdf – resume of blif, not an academic paper. (2005)
4. Biere, A.: The aiger and-inverter graph (aig) format (2007). URL http://fmv.jku.at/aiger/
5. Chakraborty, S., Gupta, A., Jain, R.: Matching multiplications in bit-vector formulas. CoRR abs/1611.10146 (2016). URL http://arxiv.org/abs/1611.10146
6. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: Performance evaluation of the vf graph matching algorithm. In: Proceedings 10th International Conference on Image Analysis and Processing, pp. 1172–1177 (1999). DOI 10.1109/ICIAP.1999.797762
7. Corno, F., Reorda, M., Squillero, G.: Rt-level itc’99 benchmarks and first atpg results. Design Test of Computers, IEEE 17(3), 44–53 (2000). DOI 10.1109/54.807894
8. Falco, V.: Vtfib: A multipurpose library built on juce. https://github.com/vinniefalco/VFLib (2008). An implementation of VFLib available at github
9. F.Brglez, Fujiwara, H.: A neutral net list of 10 combinatorial benchmark circuits and a target translator in fortran. In: International Symposium on Circuits and Systems, pp. 695–698 (1985)
10. Fu, Z., Malik, S.: Extracting logic circuit structure from conjunctive normal form descriptions. In: 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID’07), pp. 37–42 (2007). DOI 10.1109/VLSID.2007.81
11. Goldberg, E., Novikov, Y.: Berkmin: A fast and robust sat-solver. In: Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition, pp. 142–149 (2002). DOI 10.1109/DATE.2002.983269
12. Johnson, D.J., Trick, M.A. (eds.): Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, Workshop, October 11-13, 1993. American Mathematical Society, Boston, MA, USA (1996)
13. Keshavarz, S., Schellenberg, F., Richte, B., Paar, C., Holcomb, D.: Sat-based reverse engineering of gate-level schematics using fault injection and probing. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 215–220 (2018). DOI 10.1109/HST.2018.8383918
14. Li, C.M.: Equivalency reasoning to solve a class of hard sat problems. Information Processing Letters 76(1), 75–81 (2000). DOI http://dx.doi.org/10.1016/S0020-0190(00)00126-5
15. Liu, D., Yu, C., Zhang, X., Holcomb, D.: Oracle-guided incremental sat solving to reverse engineer camouflaged logic circuits. In: 2016 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 433–438 (2016)
16. Lu, F., Wang, L.C., Cheng, K.T., Huang, R.C.Y.: A circuit sat solver with signal correlation guided learning. In: 2003 Design, Automation and Test in Europe Conference and Exhibition, pp. 892–897 (2003). DOI 10.1109/DATE.2003.1253719
17. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient sat solver. In: Proceedings of the 38th Annual Design Automation Conference, DAC ’01, pp. 530–535. ACM, New York, NY, USA (2001). DOI 10.1145/378239.379017
18. Ostrowski, R., Grégoire, É., Mazure, B., Saïs, L.: Recovering and Exploiting Structural Knowledge from CNF Formulas, pp. 185–199. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
19. Roy, J.A., Markov, I.L., Bertacco, V.: Restoring circuit structure from sat instances (2004)
20. Seltzer, H.: Extracting hardware circuits from cnf formulas. Master’s thesis, Johannes Kepler Universitt Linz - Faculty of Engineering and Natural Sciences, Altenberger Straße 69 (2014)
21. Silva, J.a.P.M., Sakallah, K.A.: Grasp - a new search algorithm for satisfiability. In: Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design, ICCAD ’96, pp. 220–227. IEEE Computer Society, Washington, DC, USA (1996)
22. Silva, L.M., Andrade, F.V., Fernandes, A.O., Vieira, L.F.M.: Arithmetic circuit classification using convolutional neural networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–7 (2018). DOI 10.1109/IJCNN.2018.8489382
23. Silva, L.M., Andrade, F.V., Vieira, L.F.M.: A continuous restricted boltzmann machine and logistic regression framework for circuit classification. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020, pp. 1–8. IEEE (2020). DOI 10.1109/IJCNN48605.2020.9207323
24. Tan, Q., Potluri, S., Aysu, A.: Efficacy of satisfiability-based attacks in the presence of circuit reverse-engineering errors. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). DOI 10.1109/ISCAS45731.2020.9180705
25. Tseitin, G.: On the complexity of proofs in propositional logics. In: Seminars in Mathematics, vol. 8, pp. 466–483 (1970)
26. Yu, C., Zhang, X., Liu, D., Ciesielski, M., Holcomb, D.: Incremental sat-based reverse engineering of camouflaged logic circuits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36(10), 1647–1659 (2017). DOI 10.1109/TCAD.2017.2652220