A Two-Step Search Engine For Large Scale Boolean Matching Under NP3 Equivalence

Chak-Wa Pui , Peishan Tu , Haocheng Li , Gengjie Chen , and Evangeline F. Y. Young

Department of Computer Science and Engineering,
The Chinese University of Hong Kong, NT, Hong Kong
{cwpui,pstu,hcli,gjchen,fyyoung}@cse.cuhk.edu.hk

***Abstract*—Boolean matching is one of the most widely used engines in industrial applications. However, existing Boolean matching researches mainly focus on NPNP-equivalence. In this paper, we study a more practical problem of Boolean matching, which is Non-exact Projective NPNP (NP3). A two-step search engine is used to solve the problem and several heuristics and constraints are proposed to accelerate the whole process. In particular, we explore a new kind of symmetry properties in NP3 equivalence checking which helps to prune the solution space efficiently. Experimental results show that our proposed approach can achieve the best results among the winning teams of the ICCAD 2016 contest in quality within a given time limit.**

I. Introduction

Boolean matching is the problem of determining whether two Boolean functions are functionally equivalent under some constraints. In general, Boolean matching refers to the Boolean matching problem under NPNP-equivalence (Negation and Permutation of outputs and Negation and Permutation of inputs). However, due to the actual demands from industry, variants of the problem have been introduced and explored, such as PP-equivalence, NPN-equivalence, etc. The matching problem that we discuss in this paper is NP3-equivalence (Non-exact Projective NPNP) checking, which is an even more general formulation of the Boolean matching problem. The matching problem is formulated as an optimization problem instead of a decision problem and the objective is to obtain the maximum number of equivalent outputs instead of matching all of them. The formal definition of this problem will be given in Section II-A. This problem has much larger solution space compared to NPNP-equivalence checking, which is already proved to be between coNP and $\sum_2^p$ in the polynomial hierarchy [5], [7].

Boolean matching finds numerous applications in verification and logic synthesis. In library binding [6], Boolean matching is used to recognize whether a cell can implement a portion of the network. In engineering change order (ECO), large isomorphic sub-circuits exist in the original and slightly modified circuits [15]. Boolean matching can be used to find the minimal set of changes in logic such that large isomorphic sub-circuits can be reutilized to save designers' time and money [11]. In hardware Trojan detection, Boolean matching is used to identify candidate cones in the stored library [19]. Boolean matching is also widely used in logic verification [17] and technology mapping [8].

A. Previous Work

Prior works in Boolean matching can be classified into three major categories: signature based, canonical form based and SAT based. The goal of signature based matching [1], [6] is to prune the Boolean matching space by filtering impossible I/O correspondences. Although it is effective, most of them are incomplete due to their intrinsic limitations [16]. Canonical form based methods try to compare the canonical representations of two Boolean functions to find valid I/O matches [2]–[4], [9]. However, the scalability of this kind of methods becomes the bottleneck when solving large scale Boolean matching. Recently, several SAT-based approaches are proposed, which yields good scalability and high efficiency. In [10], a hybrid method based on SAT and signature is proposed. In [13], [14], an SAT-based Boolean matching procedure with learning is introduced. A recent work [20] on NP3 equivalence checking claims that they can obtain results of output pairs whose support sizes are small by using support set theory and symmetric detection.

However, very few of these prior works can check NPNP-equivalence or even PP-equivalence. Since the NP3 equivalence problem is formulated as an optimization problem instead of a decision problem and the solution space is much larger than that of NPNP-equivalence, a scalable Boolean matching algorithm is needed.

~~B. Applications~~

Since NP3 is a more general formulation, it can also be used to solve other Boolean matching problems like NPNP, NPN. Moreover, NP3 can find applications in ECO, hardware security, etc. For example, in ECO, given a sub-circuit in a netlist, we need to implement a new sub-circuit with fewer or more inputs or outputs. Traditionally, the sub-circuit will be modified incrementally to meet the new requirement. By using NP3 checker, we can determine directly if two sub-circuits are equivalent by multiple binding of inputs and binding inputs to constants. In hardware security, identifying a circuit as a Trojan is very time consuming. By finding the maximum numbers of matched inputs and outputs of the two circuits with an NP3 checker, the candidate circuits can be filtered such that only those that are likely to be Trojans will be testified. Moreover, for some multi-usage IP cores, the number of inputs might be more than the required number of inputs, which gives the opportunity to hardware Trojan attacks. If certain signals are given to these "redundant" inputs, the IP core might become a Trojan. By using NP3 checker, we can check if a multi-usage IP core is equivalent to a known Trojan of the current application, since NP3 matching allows multiple binding of inputs and binding inputs to constants.

C. Our Contributions

In this paper, we propose a search engine for large scale Boolean matching under NP3 equivalence. The major contributions are summarized as follows:

- A two-step search engine to solve large scale Boolean matching under NP3 equivalence is proposed. This two-step approach enables us to solve the optimization problem more efficiently and effectively. To the best of our knowledge, this is the first work that gives a systematic method to solve large scale Boolean matching under NP3 equivalence.

978-1-5090-0602-1/18/$31.00 ©2018 IEEE

- Several heuristics are used to accelerate the searching process, which include modifying the matching order of output pairs, output grouping, and output group signature. The first two help the output solver to find a valid output matching faster and avoid bad matching results in the searching process. The output group signature heuristics relaxes the constraint under NPNP to handle projection and constant binding.
- New constraints are proposed to solve the Boolean matching problem under NP3 equivalence, which include support group size dependency constraints and symmetry related constraints. The support group size dependency constraints explore the relationship between the functional support sizes of the two given circuits. The symmetry related constraints explore new kinds of symmetry properties in NP3 equivalence which are different from those explored in previous works on NPNP equivalence, PP equivalence, etc.

The remainder of this paper is organized as follows. Section II gives the preliminaries of the problem. Section III–V first gives an overview of our algorithm and then introduces its details. Section VI shows the experimental results, and we finally conclude in Section VII.

II. Preliminaries

In this paper, a set of Boolean variables is denoted as an upper-case letter, e.g. X; its elements are in lower-case letters, e.g., $x_i \in X$; an instance of X is denoted as $\vec{x}$; an instance of X where x_i (x_j) is set to p (q) is denoted as $\vec{x}|_{x_i=p,x_j=q}$. The cardinality of a set X is denoted as $|X|$. Given a function $f(X)$, its outputs are denoted as $< f_1, f_2, \ldots, f_{|f|} >$, where $|f|$ is the number of outputs of f. An instance of $f(X)$ is denoted as $\vec{f}(\vec{x})$.

Given two circuits *ckt0* and *ckt1*, they represent two functions $f(X)$ and $g(Y)$ respectively. $|X|$ denotes the number of primary inputs of *ckt0* and $|f|$ represents the number of primary outputs of *ckt0*. Similarly in *ckt1*, $|Y|$ and $|g|$ represent the number of primary inputs and outputs respectively. For simplicity, we assume $|X| = m_I$, $|f| = m_O$, $|Y| = n_I$, $|g| = n_O$ in the remaining parts of the paper.

A. Boolean Matching

A permutation ψ from X to Y is a bijection function $\psi : X \to Y$. A phase assignment ϕ over X is a component-wise mapping with $\phi(x_i) = x_i$ or $\neg x_i$.

Given two target functions $f(X)$ and $g(Y)$ where $m_I = n_I, m_O = n_O$, Boolean matching under NPNP equivalence is to check the equivalence of these two functions under input/output permutation and phase assignment. In other words, we need to check if there exists a set of ψ_I, ψ_O, ϕ_I, ϕ_O such that $\phi_O \circ \psi_O(\vec{f}(\vec{x})) = \vec{g}(\phi_I \circ \psi_I(\vec{x}))$ for any $\vec{x}$.

In the following, we are going to define NP3 equivalence. For two target functions $f(X)$ and $g(Y)$, we can construct two 0-1 matrices M_I and M_O as below.

$$M_I = \begin{array}{c} \\ y_1 \\ y_2 \\ \vdots \\ y_{n_I} \end{array} \begin{array}{c} \begin{array}{cccccccc} x_1 & \neg x_1 & \cdots & x_{m_I} & \neg x_{m_I} & 0 & 1 \end{array} \\ \begin{bmatrix} a_{1,1} & b_{1,1} & \cdots & a_{1,m_I} & b_{1,m_I} & a_{1,m_I+1} & b_{1,m_I+1} \\ a_{2,1} & b_{2,1} & \cdots & a_{2,m_I} & b_{2,m_I} & a_{2,m_I+1} & b_{2,m_I+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ a_{n_I,1} & b_{n_I,1} & \cdots & a_{n_I,m_I} & b_{n_I,m_I} & a_{n_I,m_I+1} & b_{n_I,m_I+1} \end{bmatrix} \end{array},$$

$$M_O = \begin{array}{c} \\ g_1 \\ g_2 \\ \vdots \\ g_{n_O} \end{array} \begin{array}{c} \begin{array}{cccccc} f_1 & \neg f_1 & \cdots & f_{m_O} & \neg f_{m_O} \end{array} \\ \begin{bmatrix} c_{1,1} & d_{1,1} & \cdots & c_{1,m_O} & d_{1,m_O} \\ c_{2,1} & d_{2,1} & \cdots & c_{2,m_O} & d_{2,m_O} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ c_{n_O,1} & d_{n_O,1} & \cdots & c_{n_O,m_O} & d_{n_O,m_O} \end{bmatrix} \end{array}.$$

M_I shows how the inputs of f are mapped to the inputs of g, where $a_{i,j} = 1$ ($b_{i,j} = 1$) denotes that the positive (negative) of x_j is mapped to y_i. Noted that, $a_{i,m_{I+1}} = 1$ ($b_{i,m_{I+1}} = 1$) denotes y_i is mapped to constant 0 (1). In M_O, $c_{i,j} = 1$ ($d_{i,j} = 1$) denotes that the positive (negative) of f_j is mapped to g_i. In non-exact projective NPNP (NP3) Boolean matching, $f(X)$ and $g(Y)$ are equivalent if there exists an assignment of M_I, M_O such that the constraints in Equation (1)–(5) are satisfied.

$$\sum_{i=1}^{n_O}\sum_{j=1}^{m_O}(c_{i,j} + d_{i,j}) > 0, \tag{1}$$

$$\sum_{j=1}^{m_O}(c_{i,j} + d_{i,j}) \leq 1, \quad \forall i = 1, \cdots, n_O, \tag{2}$$

$$\sum_{j=1}^{m_I+1}(a_{i,j} + b_{i,j}) = 1, \quad \forall i = 1, \cdots, n_I, \tag{3}$$

$$(\bigwedge_{c_{j,i}=1} f_i \equiv g_j) \wedge (\bigwedge_{d_{j,i}=1} f_i \equiv \neg g_j), \tag{4}$$

$$(\bigwedge_{a_{j,i}=1} x_i \equiv y_j) \wedge (\bigwedge_{b_{j,i}=1} x_i \equiv \neg y_j). \tag{5}$$

By Equation (1), at least a pair of outputs should be matched. Equation (2) ensures an output of g can be mapped to at most one output of f while Equation (3) ensures every input of g should be mapped to either an input of f or a constant. The last two constraints make sure the inputs and outputs of f and g are bound as M_O and M_I indicate.

The key differences between NP3 and other Boolean equivalence problem are summarized as follows:

- Some outputs and inputs of function f can be unmatched.
- Inputs of g can be bound to constant.
- Multiple inputs (outputs) of g can be bound to the same input (output) of f.

B. Problem Definition

Given two circuits *ckt0* and *ckt1* whose target functions are $f(X)$ and $g(Y)$, if they are under NP3 equivalence, we can find an assignment of M_I, M_O satisfying Equation (1)–(5). To determine how good a given Boolean matching under NP3 equivalence is, Equation (6) is used to evaluate the result and higher score indicates better matching result. This evaluation method aims at maximizing the number of mapped outputs of both $ckt0$ and $ckt1$.

$$\text{score} = \sum_{i=0}^{m_o} q(f_i), \tag{6}$$

where f_i denote the ith primary output of *ckt0* and $q(fi)$ is calculated as Equation (7).

$$q(f_i) = \begin{cases} K + \sum_{j=1}^{n_O}(c_{j,i} + d_{j,i}), & \text{if } \sum_{j=1}^{n_O}(c_{j,i} + d_{j,i}) \geq 1, \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

If K is a big number, the number of mapped outputs of $ckt0$ will be preferred over the total number of mapped outputs of both circuits. To be specific, if K is large, matched results like $\{(f_i, g_j), (f_p, g_q)\}$ will have higher scores than those like $\{(f_i, g_j, g_p, g_q)\}$ even though they have the same number in total matched outputs.

To determine whether a matching result is optimal, we compare it with the one maximizing the number of output matching groups and assuming that all of the remaining unmatched outputs of $ckt1$ are matched to the same output of $ckt0$. For example, given two circuits $ckt0$ and $ckt1$ whose numbers of outputs are 12 and 15 respectively, a result is considered optimal if its score is $((K + 2) \times 12 + 3)$.

Algorithm 1 SAT-based Backtracking

Require: Two functions f and g
let R be the output matching results;
while !optimal and !timeout **do**
  if new pair can be found by output SAT **then**
    add a new output matching pair to R;
  else
    if projection is disabled **then**
      enable projection;
    else
      forbid the current matching result R;
      remove the last added pair from R;
      disable projection;
    end if
    continue;
  end if
  solve R by input solver;
  if R can not be satisfied **then**
    forbid the current matching result R;
    remove the last added pair from R;
  end if
end while
return R;

III. Overview

Our Boolean matching engine is a two-step search engine based on backtracking and SAT. The output solver is a combination of SAT and a backtracking search while the input solver is based on the framework of [14]. Given two functions, we first construct two And-Invert Graphs (AIGs) [12]. The output solver will then generate some output matching pairs. After simplifying the AIGs, the input solver will try to get an input matching by another search such that the output matching is satisfied under NP3 equivalence. The result from the input solver will give feedback to the output solver, such that the output solver can better optimize the matching result. The details of our output solver and input solver will be discussed in Section IV and Section V respectively. In the following sections, if a method is stated as a "heuristic", sub-optimality may be resulted. Otherwise, the method described will not lead to loss in optimality.

IV. Output Solver

In this section, we will first introduce the basic framework of our output solver in Section IV-A. In Section IV-B, three constraints considering the output matching from a functional perspective are discussed. Two heuristics are then proposed in Section IV-C and IV-D to improve the efficiency in output matching.

A. SAT-based Backtracking

As shown in Algorithm 1, our output solver uses SAT-based backtracking search and it optimizes the matching result iteratively by using the feedback from the input solver. When an output matching is proved to be feasible under NP3 equivalence by the input solver, it will keep those output matching pairs in the next iteration. If it is proved to be not feasible, the output solver will avoid selecting this output matching again in later iterations. When the output SAT solver returns no solution, meaning that no more output matching result can be found if the previous output matching pairs are kept, backtracking will then be performed.

In Boolean matching under NP3 equivalence, multiple g_j can be mapped to f_i, and this feature is called projection. If g_1 and g_2 are projected to f_1, we have two output matching pairs, i.e. f_1 to g_1 and f_1 to g_2. Since each g_j can only be mapped to one f_i, allowing projection at the beginning will lower the flexibility of later iterations in output matching, resulting in poor performance. To avoid this, projection is not allowed until backtracking is performed. As shown in Equation (8), the variable *allowProj* is used to decide whether projection is allowed in the output solver.

$$\bigwedge_{i=1}^{n_O}\bigwedge_{j=i+1}^{n_O}\bigwedge_{k=1}^{m_O}(\textit{allowProj}\vee\neg c_{i,k}\vee\neg c_{j,k})\wedge(\textit{allowProj}\vee\neg c_{i,k}\vee\neg d_{j,k}) \wedge(\textit{allowProj}\vee\neg d_{i,k}\vee\neg c_{j,k})\wedge(\textit{allowProj}\vee\neg d_{i,k}\vee\neg d_{j,k}) \tag{8}$$

When *allowProj* is set to 1, Equation (8) is always true. Otherwise, at least one of $(c_{i,k}\vee d_{i,k})$ and $(c_{j,k}\vee d_{j,k})$ must be zero and no projection is thus allowed in output matching.

B. Output Functional Constraints

Definition 1. *Given a function $f(X)$, x_i and f_i are **functional support** of each other if there exists a $\vec{x}$ such that flipping the value of x_i in $\vec{x}$ changes the value of f_i. FuncSupp(f_i) (FuncSupp(x_i)) denotes the set of functional support of f_i (x_i).*

Definition 2. *Given a function $f(X)$, x_i and f_i are **structural support** of each other if there is a path between x_i and f_i in the corresponding AIG. StrucSupp(f_i) (StrucSupp(x_i)) denotes the set of structural support of f_i (x_i).*

Definition 3. *Given a function $f(X)$, the **fanouts (fanins)** of x_i (f_i) are the set of gates connected to x_i (f_i) in the corresponding AIG.*

Given two output f_i and g_j, if |FuncSupp(f_i)| is bigger than |FuncSupp(g_j)|, these two outputs cannot form a valid pair under NP3 equivalence for any input matching. These can be further extended to multiple output matching pairs. Given an output matching, any valid input matching result under NP3 equivalence must satisfy,

$$|\bigcup_{f_i\in f_M}\text{FuncSupp}(f_i)|\le|\bigcup_{g_j\in g_M}\text{FuncSupp}(g_j)|, \tag{9}$$

where f_M, g_M denote groups of matched outputs. As experimental results show, most of the time in the searching process is spent by the input solver. It is worthwhile to check whether the output matching to be tried violates this constraint.

Definition 4. *For an output $f_i \in f$, **Equal**(f_i) denotes all the $f_j \in f$ that have the same value of f_i for any input $\vec{x}$.*

Given the AIG of a function f, we can easily check whether f_i and f_j share the same fanin signals, which implies $f_i = f_j$ or $f_i = \neg f_j$ for any $\vec{x}$. There are five cases for any two output matching pairs (f_p, g_q) and (f_i, g_j) as shown below,

- $g_q \in \text{Equal}(g_j) \wedge f_p \in \text{Equal}(f_i)$,
- $g_q \in \text{Equal}(g_j) \wedge \neg f_p \in \text{Equal}(f_i)$,
- $\neg g_q \in \text{Equal}(g_j) \wedge \neg f_p \in \text{Equal}(f_i)$,
- $\neg g_q \in \text{Equal}(g_j) \wedge f_p \in \text{Equal}(f_i)$,
- otherwise.

If the matching result belongs to one of the first four situations, some constraints must be satisfied as discussed in the followings. For simplicity, we only discuss one of the situations, the other three are similar. If $g_q \in \text{Equal}(g_j)$ and $f_p \in \text{Equal}(f_i)$, the constraint in Equation (10) should not be violated.

$$(\neg c_{j,i}\vee c_{q,p})\wedge(c_{j,i}\vee\neg c_{q,p})\wedge(\neg d_{j,i}\vee d_{q,p})\wedge(d_{j,i}\vee\neg d_{q,p}), \tag{10}$$

where $(\neg c_{j,i}\vee c_{q,p})\wedge(c_{j,i}\vee\neg c_{q,p})$ means that if g_j is matched to f_i, g_q should also be matched to f_p, and vice versa. Similar meaning can be applied to $(\neg d_{j,i}\vee d_{q,p})\wedge(d_{j,i}\vee\neg d_{q,p})$.

Algorithm 2 Generate output groups of f, g

Require: Two functions f and g, $|f| = |g|$
1: $n \leftarrow |f|$;
2: $f^s, g^s \leftarrow$ sorted outputs of f, g as in Section IV-C;
3: add a new group to G_f and G_g respectively;
4: **for** $i = n : 1$ **do**
5: add f_i^s to the last group in G_f;
6: add g_i^s to the last group in G_g;
7: **if** $|\text{FuncSupp}(f_i^s)| > |\text{FuncSupp}(g_{i-1}^s)|$ **then**
8: add a new group to G_f and G_g respectively;
9: **end if**
10: **end for**
11: **return** G_f, G_g;

C. Output Matching Order Heuristics

Given two functions f and g, f_i is more likely to be mapped to g_j if they are similar to each other both functionally and structurally. Moreover, we prefer to match those outputs that have smaller number of functional supports first since they can give more feedback to later iterations and yield better performance in terms of running time. To improve the efficiency of our algorithm, we use a heuristics to adjust the output matching order. f and g are first sorted by their functional support sizes, structural support sizes and fanin sizes in ascending order, where functional support size is used as the first order while structural support size and fanin size serve as the second and third. The variables in M_O are then created in this order.

D. Output Grouping Heuristics

Consider two functions f and g, where $|f| = |g| = 4$, and the numbers of functional supports of f_1, f_2, f_3, f_4 are $1, 2, 3, 5$, and the numbers of functional supports of g_1, g_2, g_3, g_4 are $2, 2, 4, 6$. Due to the constraint in Equation (9), g_1 and g_2 can only be matched to f_1 or f_2, g_3 can only be matched to f_1, f_2 or f_3, g_4 can be matched to any one of f_1, f_2, f_3 and f_4. Therefore, if f_1 is matched to g_4 at the beginning, either f_3 or f_4 cannot be matched to any g_i. To avoid this situation, a heuristics is used. As mentioned in Section IV-B, an output $f_i \in f$ can be matched if there exists an output $g_j \in g$ such that $|\text{FuncSupp}(f_i)| \leq |\text{FuncSupp}(g_j)|$. After removing all the outputs that cannot be matched, for the functions having the same number of outputs, we first construct two set G_f, G_g of output groups as shown Algorithm 2. The output groups of the previous example will be $\{g_1, g_2\}, \{g_3\}, \{g_4\}$ and $\{f_1, f_2\}, \{f_3\}, \{f_4\}$. Then, we can get the group number of each output, e.g. the group numbers for f_1, f_2, f_3, f_4 (g_1, g_2, g_3, g_4) are $1, 1, 2, 3$ $(1, 1, 2, 3)$ in the previous example. It is easy to see that matching a pair of outputs with different group numbers will result in at least a pair of outputs not being matched. In order to maximize the number of output matching pairs, we will not match outputs of different group numbers in this heuristics if the numbers of outputs that can be matched in f and g are the same.

V. Input Solver

In this section, we will first introduce the basic framework of our input solver in Section V-A. In Section V-B, some useful constraints considering the input matching from a functional perspective are proposed. A heuristics shown in Section V-C is then used to improve the input matching efficiency. Finally, we explore the symmetry properties in NP3 equivalence and propose some efficient constraints in Section V-D to prune the solution space effectively.

Algorithm 3 Get the redundant set R_{f_i} of $\vec{x}$ on f_i

Require: a function f_i and an input instance $\vec{x}$;
1: $R_{f_i} \leftarrow \emptyset$;
2: **for** $p = 1, \ldots, m_I$ **do**
3: add x_p to R_{f_i};
4: $\forall x_q \notin R_{f_i}$, x_q is bound to $\vec{x}_q$;
5: **if** $\exists x_q \in R_{f_i}$ is not redundant **then**
6: remove x_p from R_{f_i};
7: **end if**
8: unbind all the inputs of f_i
9: **end for**
10: **return** R_{f_i};

A. Incremental SAT-based Solver

Our input solver is based on the framework mentioned in [13], [14]. It is an incremental SAT-based solver with strengthened learning. MiniSat [18] is used to get a solution of M_I and M_O under the constraints of NP3 equivalence and those new constraints we propose (in Section V-B–V-D). The mapping solution is then examined by solving the miter shown in Equation (11).

$$\psi = \sigma \wedge \bigvee_{c_{j,i}+d_{j,i}=1} ((f_i(X) \equiv f_i) \wedge (g_j(Y) \equiv g_j) \wedge (\phi_O(f_i) \equiv g_j)), \tag{11}$$

where $f_i(X)$ $(g_j(Y))$ denotes the function determining the value of f_i (g_j) and

$$\sigma = (\bigwedge_{a_{j,i}=1} (x_i \equiv y_j)) \wedge (\bigwedge_{b_{j,i}=1} (x_i \equiv \neg y_j)) \wedge (\bigwedge_{a_{j,m_I+1}=1} (y_j \equiv 0)) \wedge (\bigwedge_{b_{j,m_I+1}=1} (y_j \equiv 1)).$$

If the solver returns a solution, this solution is a counter example to the matching result. Otherwise, a valid input matching under NP3 equivalence is found. These steps are repeated until a mapping solution satisfying the NP3 equivalence constraints is found or the number of trials exceeds a limit, which is proportional to the number of gates in the AIG in our implementation.

Theorem 1. *Given f_p and g_q for Boolean matching under NP3 equivalence, if $f_p(\vec{x}) \neq g_q(\vec{y})$, then any assignment of M_I is infeasible if it maps $\vec{x}$ to $\vec{y}$.*

(The proof of Theorem 1 is skipped due to lack of space.)

With Theorem 1, by excluding some infeasible assignments of M_I, a counter example can be used to further prune the solution space when solving the SAT to get the next M_I. To achieve that, Equation (12) is added to the input solver SAT, where $\vec{x}$ and $\vec{y}$ are a counter example of an output matching pairs (f_p, g_q).

$$(\bigvee_{i=1}^{n_I} \bigvee_{j=1}^{m_I} l_{i,j}) \wedge (\bigvee_{i=1}^{n_I} c_i), \tag{12}$$

where

$$l_{i,j} = \begin{cases} b_{i,j}, & \text{if } \vec{x}_j = \vec{y}_i, \\ a_{i,j}, & \text{if } \vec{x}_j \neq \vec{y}_i, \end{cases}, c_i = \begin{cases} a_{i,m_I+1}, & \text{if } \vec{y}_i = 1, \\ b_{i,m_I+1}, & \text{if } \vec{y}_i = 0. \end{cases}$$

Definition 5. *For a counter example $\vec{x}$ on (f_p, g_q), the **redundant set** R_{f_p} is a set of inputs of f_p that will not change the value of f_p if the other inputs not in the set are bound to $\vec{x}$.*

In fact, more counter examples can be derived from the current counter examples without running the input solver. For example, given a counter example of $\vec{x} = 1101$, if $R_{f_p} = \{x_1, x_2\}$, we

can get three more counter examples 1001, 0101, and 0001. In our solver, based on Equation (12), the solution space of M_I will be further pruned by this fact. Given a counter example $\vec{x}$ and $\vec{y}$ such that $f_p(\vec{x}) \neq g_q(\vec{y})$, we first use a greedy method shown in Algorithm 3 to get its redundant set. Since changing the values of the $x_i \in R_{f_p}$ will not change the value of f_p, Equation (12) can be modified to Equation (13), which has less literals.

$$(\bigvee_{y_i \notin R_{g_q}} \bigvee_{x_j \notin R_{f_p}} l_{i,j}) \wedge (\bigvee_{y_i \notin R_{g_q}} c_i). \tag{13}$$

It is obvious that the less literals in Equation (13) the more infeasible solutions will be pruned. For a counter example $\vec{x}$ and $\vec{y}$ on multiple outputs, we will select the R_{f_p} whose size is the largest on its corresponding output. Given a matched output pair (f_p, g_q), if $|\text{FuncSupp}(f_p)| = |\text{FuncSupp}(g_q)|$, no input $y_j \in \text{FuncSupp}(g_q)$ can be bound to a constant. Hence, c_i will not be added if the corresponding y_i cannot be bound to a constant.

Moreover, since the clauses added by Equation (13) can be reused when solving subsequent output matching results, we will add these clauses to the input SAT solver if their corresponding output matching pairs are a subset of the current output matching. By reusing these clauses in the input solver, we can avoid redundant search.

B. Input Functional Constraints

After getting the output matching result, we can simplify the AIGs of $f(X)$ and $g(Y)$ by removing those gates that are not connected to any f_i or g_j in the matching pairs. Before introducing the input functional constraints, we first need to define floating variable as Definition 6.

Definition 6. ***Floating variables*** *are denoted as f_F, g_F, X_F, Y_F and should satisfy the following constraints.*

- $f_F = \{f_j | \sum_{i=1}^{n_O} c_{i,j} + d_{i,j} = 0\}$,
- $g_F = \{g_i | \sum_{j=1}^{m_O} c_{i,j} + d_{i,j} = 0\}$,
- $X_F = \{x_i | x_i \notin \bigcup_{f_i \notin f_F} \textit{FuncSupp}(f_j)\}$,
- $Y_F = \{y_i | y_i \notin \bigcup_{g_j \notin g_F} \textit{FuncSupp}(g_j)\}$.

These variables are irrelevant to the current input matching problem since they are either unmapped outputs or inputs not supporting any mapped outputs. To accelerate the input matching process, we map the floating variables $y_i \in Y_F$ to constant 0 while $x_p \in X_F$ cannot be mapped to any $y_j \in Y$. For any $x_i \in X$ that is functionally supported by some non-floating variables, it can only be mapped to a y_j that satisfies Equation (14).

$$y_j \in \bigcup_{f_p \in \text{FuncSupp}(x_i)} \bigcup_{c_{q,p}+d_{q,p}=1} \text{FuncSupp}(g_q). \tag{14}$$

C. Output Group Signature Heuristics

As mentioned in Section IV-D, the heuristics will forbid the outputs in different groups to be matched if the numbers of outputs that can be matched in f and g are the same. Given two inputs x_i and y_j, W_{x_i} (W_{y_j}) denotes the set of groups that x_i (y_j) supports. It is easy to deduce from Equation (14) that any valid input matching x_i and y_j must satisfy Equation (15), which can be used to prune the solution space of M_I.

$$W_{x_i} = W_{y_j}. \tag{15}$$

However, since projection and constant binding are allowed under NP3 equivalence, the output groups that y_j supports may change under different input matching, which cannot be detected until an input matching is selected. For example, y_0 may support three outputs g_1, g_2 and g_3 which belong to group 1, 2, and 2 respectively. Assume that y_2 is also a functional support of g_1 and is now bind to a constant, which may make y_0 redundant to g_1, thus y_0 no longer supports group 1.

There are three situations: (1) $W_{x_i} = W_{y_j}$, (2) $W_{x_i} \subset W_{y_j}$, (3) $W_{x_i} \not\subset W_{y_j}$. It's obvious that x_i and y_j will not be matched in situation (3) under any valid input matching results. If W_{x_i} is a proper subset of W_{y_j}, it is still possible for x_i to be matched to y_j since y_j may not support some of the groups under some input matching. A heuristics is proposed to handle situation (2), which aims at identifying those input matching pairs (x_i, y_j) that are very likely to violate Equation (15) under NP3. Let w be the group that x_i does not support but y_j supports ($w \in W_{y_j}$ and $w \notin W_{x_i}$). The heuristics only allows two inputs to be matched if the following two conditions are satisfied.

- The number of outputs of g in w is less than a constant, which is set to be 3 in our implementation. This constraint helps to find the output groups that y_j might not support under some input matching because the number of outputs that y_j supports in those groups are relatively small.
- Since only projection and constant binding can remove an input from the functional supports of an output, for any $g_q \in w$, there must be projection or constant binding in any output matching (f_p, g_q); otherwise y_j will always support w. Hence, if f_p and g_q are matched, $|\text{FuncSupp}(f_p)| < |\text{FuncSupp}(g_q)|$.

D. Symmetry Related Constraints

In this section, we study some symmetry related constraints and define negative and positive symmetry as follows:

Definition 7. *A pair of input (x_i,x_j) is **positive symmetric** on f_p if $f_p(\vec{x}|_{x_i=0,x_j=1}) = f_p(\vec{x}|_{x_i=1,x_j=0})$ for any $\vec{x}$. It is **negative symmetric** on f_n if $f_n(\vec{x}|_{x_i=0,x_j=0}) = f_n(\vec{x}|_{x_i=1,x_j=1})$ for any $\vec{x}$. A pair of input (x_i,x_j) is **symmetric** if it is positive or negative symmetric.*

In Boolean matching under NP3 equivalence, there are two kinds of symmetric properties: (1) symmetry under NP, (2) symmetry under NP3. In NP3 equivalence checking, y_i can be mapped to a constant and multiple y_j can be mapped to a x_i. The symmetric property under this kind of matching is called "symmetry under NP3". If the functional support sizes of the outputs in an output matching pair (f_i, g_j) are the same, no input of f_i is matched to multiple inputs of g_j and the inputs of g_j are not bound to any constants, finding input matching of this pair of output can be reduced to NP Boolean matching. We call the symmetric property under this kind of matching "symmetry under NP". Only positive symmetry is considered in the following discussion, the conditions that include negative symmetry can be deduced accordingly. The proof of Theorem 2 and 3 will be omitted due to page limit.

1) Symmetry under NP3: For symmetry under NP3, since multiple y_j can be mapped to one x_i and y_i can be mapped to a constant, some of the inputs may become redundant to some non-floating outputs, which will change some symmetric properties of the inputs. For example, given $g_1 = (y_1 \bigoplus y_2) \wedge y_3$, if we bind y_1 and y_2 to the same constant, y_3 will become redundant to g_1. However, this cannot be detected before an input matching is obtained.

Theorem 2. *Given a function φ ($|\varphi| = 1$), a positive symmetric group is an input group whose elements are all positive symmetric to each other. If one of the elements inside the group becomes redundant, then all the other elements in the group become redundant.*

For example, given a pair of positive symmetric inputs (y_1, y_2) of g_1, $g_1(\vec{y}|_{y_1=1,y_2=0}) = g_1(\vec{y}|_{y_1=0,y_2=1})$. If y_1 is redun-

dant, we can show that $g_1(\vec{y}|_{y_1=1,y_2=0}) = g_1(\vec{y}|_{y_1=0,y_2=0})$ and $g_1(\vec{y}|_{y_1=0,y_2=1}) = g_1(\vec{y}|_{y_1=1,y_2=1})$. Hence, y_2 must also be redundant.

Definition 8. *Given a positive symmetric group w of g_j and a pair of output matching (f_i, g_j), if*

$$|FuncSupp(g_j)| - |w| < |FuncSupp(f_i)|,$$

*w is a **hard symmetric group** of g_j under the output matching pair (f_i, g_j).*

Theorem 3. *Given a hard symmetric group w of g_j under the output matching pair (f_i, g_j), for any $y_p, y_q \in w$, they are symmetric to each other unless one of them is bound to a constant or an input of g_j.*

For example, assume g_1 is matched to f_1 and $w = (y_1, y_2)$ is a hard symmetric group of g_1 such that $|\text{FuncSupp}(g_1)| - |\text{FuncSupp}(f_1)| = 1$. If neither y_1 nor y_2 is bound to a constant or an input of g_j, it is easy to see that the symmetric property between (y_1, y_2) can only be removed when either of them becomes redundant. By Theorem 2, if y_1 is redundant, y_2 must also be redundant, and vice versa. This will make the number of functional supports of g_1 smaller than that of f_1. By Equation (9), f_1 and g_1 cannot be matched. Hence, as long as g_1 is mapped to f_1, the symmetric property between (y_1, y_2) can only be removed if either of them is bound to a constant or an input of g_j.

Given an output matching pair (f_i, g_j), we have a set W of hard symmetric groups of g_j and a set H of symmetric groups of f_i. Now, we are going to make use of this symmetry property to form input matching. With Theorem 3, we can have the following constraints.

- For any $y_p \in w_m$ where $w_m \in W$, there exists a $x_q \in h_n$ where $h_n \in H$ that is mapped to y_p unless one of the following constraints are satisfied,
  - y_p is bound to a constant or an input of g_j,
  - All the other variables in its group w_m are bound to constants or inputs of g_j.

  For example, given $W = \{\{y_1, y_2, y_3\}, \{y_4, y_5\}\}$ and $H = \{\{x_1, x_2, x_3\}\}$, y_1 have to be matched to an $x_i \in \{x_1, x_2, x_3\}$ unless either of these two conditions is satisfied: (1) it is mapped to an input of g_j or a constant; (2) y_2, y_3 are mapped to inputs of g_j or constants. Similar rules can be applied to y_2, y_3, y_4 and y_5.
- For any $y_p \in w_m$ where $w_m \in W$, if one of the variables in w_m is matched to a variable in the symmetric group $h_n \in H$, y_p will also be mapped to one of the variables in h_n otherwise y_p is bound to a constant or an input of g_j. For example, given $W = \{\{y_1, y_2, y_3\}\}$ and $H = \{\{x_1, x_2, x_3\}, \{x_4, x_5\}\}$, if there exists a $y_i \in \{y_1, y_2, y_3\}$ mapped to x_4 or x_5, each $y_i \in \{y_1, y_2, y_3\}$ should be mapped to either x_4 or x_5 otherwise it is mapped to a constant or an input of g_j.

2) Symmetry under NP: Unlike symmetry under NP3, the symmetric property under NP will remain the same under any input matching since there is no projection nor constant binding in input matching under NP equivalence. In the following, a general constraint will first be introduced, then a constraint that can be applied to some special cases will be given. First we define the symmetry signature for each input x_i of f as below:

Definition 9. *Given a function f, the symmetry signature for each input x_i, denoted as **SymmSign**$_i$, is a sequence of integers of which the jth number is denoted as SymmSign$_i(j)$, and SymmSign$_i(2j)$ (SymmSign$_i(2j+1)$) equals the number of inputs that x_i is positive (negative) symmetric to on f_j.*

For any pair of matched outputs (f_p, g_q) whose functional support sizes are the same, two inputs x_i and y_j can be matched if and only if $\text{SymmSign}_i(2p) = \text{SymmSign}_j(2q)$ and $\text{SymmSign}_i(2p+1) = \text{SymmSign}_j(2q+1)$.

For those input pairs that are positive symmetric on all the mapped outputs, we can have tighter constraints. These inputs can be divided into several positive symmetric groups such that inputs inside the same group are positive symmetric to each other. We denote the positive symmetric groups on f as $W = \{w_1, w_2, \ldots, w_{|W|}\}$ and those on g as $H = \{h_1, h_2, \ldots, h_{|H|}\}$. A group w_p is said to be mapped to h_q if all the inputs in w_p are mapped to those in h_q. It is obvious that $|W| = |H|$ otherwise a pair of symmetric inputs will be mapped to a pair of non-symmetric inputs. For any $w_p \in W$, there must exist a $h_q \in H$ that w_p is mapped to and the following constraints must be satisfied,

- $|w_p| = |h_q|$ otherwise a pair of symmetric inputs will be mapped to a pair of non-symmetric inputs,
- Since arbitrary input matching between w_p and h_q are equivalent, for all $x_i \in w_p$ and $y_i \in h_q$, x_i is mapped to y_i where $i = 1, \ldots, |w_p|$.

For example, given $W = \{\{x_1, x_2, x_3\}, \{x_4, x_5\}, \{x_6, x_7\}\}$ and $H = \{\{y_1, y_2, y_3\}, \{y_4, y_5\}, \{y_6, y_7\}\}$, $x_i \in \{x_1, x_2, x_3\}$ can only be mapped to $y_i \in \{y_1, y_2, y_3\}$ and x_4 can be mapped to either y_4 or y_6.

VI. Experimental Results

To evaluate our proposed method, the algorithms are implemented in C++. The experiments were performed on a 64-bit Linux workstation with Intel Xeon 3.7GHz CPU and 16GB memory, using the benchmarks provided by ICCAD 2016 Contest [21]. The running time limit has been set to 1800 seconds in order to have a fair comparison with the contest winners. Since we want to match as many outputs of $ckt0$ as possible, K is set to 11, which is the same as the configuration in the ICCAD 2016 contest. Note that the score and running time of each test case in the following indicate the best score the solver can achieve and the corresponding running time to get that result.

For each benchmark, before entering our flow, we will first perform a PP-equivalence checking using the method described in [10] and for those output matching pairs that are independent on others, we will perform a P-equivalence checking. If a valid solution is found, we will not perform our NP3 equivalence checking. In fact, we can only find PP-equivalence in case 13–15, and there are four pairs of outputs that are under P-equivalence in case 3.

In TABLE I, we compare the quality and running time of our Boolean matching algorithm with the winning teams in the ICCAD 2016 contest, [20], and the known optimal results of some test cases. Since the computer configuration used in [20] is different (Intel Core i5 2.9GHz CPU) and the running time of the teams are not announced, our running time is shown as a reference. For those test cases we obtain results, our solutions are optimal except for case 20 and 26. Compared to the winning teams in ICCAD 2016 contest, our framework outperforms them in all test cases. Compared to [20], our quality is almost 5X better than theirs.

In Table II, we compare the running time and quality of our framework before and after applying the symmetry constraint in Section V-D. Since only some of the test cases in the benchmark have symmetry property, we just show the comparisons for those test cases. After applying the symmetry constraints, either better quality or faster running time with the same quality can be achieved. Note that the results of case 14–15 as reported in TABLE I are obtained by PP-equivalence checking [10] but they can also be solved by our

TABLE I: Boolean matching quality and running time comparison with ICCAD 2016 contest winners.

Case#	Ours		1st Place	2nd Place	3rd Place	[20]		Optimal
	Score	Time(s)	Score	Score	Score	Score	Time(s)	
0	25	1	25	25	25	25	1	✓
1	192	18	192	192	192	48	17	✓
2	192	13	192	192	192	36	5	✓
3	180	57	180	136	180	132	9	✓
4	192	3	192	192	192	36	10	✓
5	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-
7	-	-	-	-	-	-	-	-
8	-	-	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-
10	24	1	24	24	24	24	1	✓
11	-	-	-	-	-	-	-	-
12	60	29	60	60	60	-	-	✓
13	120	718	120	-	-	-	-	✓
14	84	1	84	84	84	-	-	✓
15	120	1	120	120	120	60	2	✓
16	96	22	96	96	96	96	6	✓
17	120	3	120	120	120	-	-	✓
18	-	-	-	-	-	-	-	-
19	120	82	-	24	-	-	-	✓
20	108	479	24	48	-	12	20	-
21	-	-	-	-	-	-	-	-
22	60	24	60	48	60	-	-	✓
23	-	-	-	-	-	-	-	-
24	-	-	-	-	-	-	-	-
25	192	98	192	108	73	-	-	✓
26	120	1	120	0	0	-	-	-
Total	2005	-	1801	1469	1418	469	-	-

TABLE II: Comparison of running time and quality of our framework before and after applying symmetry constraint.

Case#	With symm		Without symm	
	Score	Time(s)	Score	Time(s)
12	60	29	-	-
14	84	324	36	1391
15	120	38	120	99
17	120	3	120	73
19	120	82	36	75
20	108	479	96	188
22	60	24	-	-

NP3 equivalence checking. The running time is longer compared with those of the method described in [10] (since only input and output permutations are considered) but are still very reasonable and affordable (as shown in TABLE II).

VII. Conclusion

Although Boolean matching is well studied in previous works, the problem of Boolean matching under NP3 equivalence has not been explored yet. It has been shown that search-based methods with certain pruning techniques work well in large scale Boolean matching. In this paper, a two-step search engine is proposed for large scale Boolean matching under NP3 Equivalence. It consist of two parts, a SAT-based backtracking output solver and an incremental SAT-based input solver. Several constraints and heuristics are used in both solvers. Experimental results shows that our framework outperforms all the winning teams in ICCAD 2016 contest. Future works include how to avoid low possibility input and output matching results, more sophisticated constraints in input solver and how to take constraints such as don't cares into consideration.

References

[1] A. Abdollahi. Signature based boolean matching in the presence of don't cares. In *ACM/IEEE Design Automation Conference (DAC)*, pages 642–647, 2008.

[2] A. Abdollahi and M. Pedram. A new canonical form for fast boolean matching in logic synthesis and verification. In *ACM/IEEE Design Automation Conference (DAC)*, pages 379–384, 2005.

[3] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto. A unified approach to canonical form-based boolean matching. In *ACM/IEEE Design Automation Conference (DAC)*, pages 841–846, 2007.

[4] G. Agosta, F. Bruschi, G. Pelosi, and D. Sciuto. A transform-parametric approach to boolean matching. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 28(6):805–817, 2009.

[5] M. Agrawal and T. Thierauf. The boolean isomorphism problem. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 422–430, 1996.

[6] L. Benini and G. De Micheli. A survey of boolean matching techniques for library binding. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2(3):193–226, 1997.

[7] B. Borchert, D. Ranjan, and F. Stephan. On the computational complexity of some classical equivalence relations on boolean functions. *Theory of Computing Systems*, 31(6):679–693, 1998.

[8] J. Cong and Y.-Y. Hwang. Boolean matching for lut-based logic blocks with applications to architecture evaluation and technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 20(9):1077–1090, 2001.

[9] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko. Fast boolean matching based on npn classification. In *FPT*, pages 310–313, 2013.

[10] H. Katebi and I. L. Markov. Large-scale boolean matching. In *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, pages 771–776, 2010.

[11] S. Krishnaswamy, H. Ren, N. Modi, and R. Puri. Deltasyn: an efficient logic difference optimizer for eco synthesis. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 789–796, 2009.

[12] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the 34th annual Design Automation Conference*, pages 263–268. ACM, 1997.

[13] C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang. Boolean matching of function vectors with strengthened learning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 596–601, 2010.

[14] C.-F. Lai, J.-H. R. Jiang, and K.-H. Wang. Boom: a decision procedure for boolean matching with abstraction and dynamic learning. In *ACM/IEEE Design Automation Conference (DAC)*, pages 499–504, 2010.

[15] A. Mishchenko, S. Ray, and R. Brayton. Incremental sequential equivalence checking and subgraph isomorphism.

[16] J. Mohnke, P. Molitor, and S. Malik. Limits of using signatures for permutation independent boolean comparison. In *ACM/IEEE Design Automation Conference (DAC)*, pages 459–464, 1995.

[17] J. Mohnke, P. Molitor, and S. Malik. Application of bdds in boolean matching techniques for formal logic combinational verification. *International Journal on Software Tools for Technology Transfer*, 3(2):207–216, 2001.

[18] N. Sorensson and N. Een. Minisat v1.13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005.

[19] P. Swierczynski, M. Fyrbiak, C. Paar, C. Huriaux, and R. Tessier. Protecting against cryptographic trojans in fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, pages 151–154, 2015.

[20] F. Wang, J. Zhang, L. Wu, W. Zhang, and G. Luo. Search Space Reduction for the Non-Exact Projective NPNP Boolean Matching Problem. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 596–601, 2017.

[21] C.-A. R. Wu, C.-J. J. Hsu, and K.-Y. Khoo. ICCAD-2016 CAD contest in non-exact projective NPNP boolean matching and benchmark suite. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, page 40, 2016.