LSV FINAL : ICCAD Boolean Matching

I. Introduction

Boolean matching is a problem to check whether two Boolean functions are functionally equivalent under some constraints. In this ICCAD contest, we define the NPNP-V2 variant from the NPNP-equivalence (Negation and Permutation of outputs and Negation and Permutation of inputs). The problem of this contest is a optimization problem and the goal is to obtain the maximum number of equivalent of inputs/outputs in 3600(s).

II. Preliminaries

A. Notation

In this report, variables of a Boolean function is denoted as an upper-case letter, e.g. X and each variable is 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 is set to p is denoted as $\vec{x}|_{x_i=p}$. The cardinality of set X is denoted as $|X|$. Given a function f(X), its outputs are denoted as $< f_1, f_2, ..., f_n >$, where n is the number of the outputs of f An instance of f(X) is denoted as $\vec{f}(\vec{x})$.

Given two circuit cir0 and cir1, they represent two Boolean function f(X) and g(Y) respectively. $|X|$ and $|f|$ are the number of primary inputs and the number of primary outputs of f respectively and same as $|Y|$ and $|g|$. For simplicity, we assume $|X| = m_i$, $|f| = m_o$, $|Y| = n_i$, and $|g| = n_o$.

B. Boolean Matching

. A permutation of 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 $\neg x_i$

Under NPNP-equivalence, the goal of Boolean matching is to find legal permutation and phase assignment. In other words, we need to find out a set of ψ_I, ψ_O, ϕ_I and ϕ_O such that $\phi_O \circ \psi_O \vec{f}(\vec{x}) = \vec{g}(\phi_I \circ \psi_I \vec{x})$

In the following, we can construct two 0-1 matrices M_I and M_O as below.

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

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

M_I shows how the inputs of f are mapped to the inputs of g. $a_{i,j} = 1$ ($b_{i,j} = 1$) means that the positive(negative) of x_j is mapped to y_i. Please note that $a_{i,m_{I+1}} = 1(b_{i,m_{I+1}} = 1)$ means that y_i is mapped to constant 0, 1. For $c_{ij}(d_{ij})$ in M_O, the definition of them are the same with M_I.

Since the number of primary outputs of two circuits are the same ($n_O = m_O$) for each test case given by the CAD contest, we only discuss the special case of NP3-equivalence [3], and define the special case as NPNP-v2 problem. In NPNP-v2 Boolean matching, $f(X)$ and $g(Y)$ are equivalent if there exists an assignment of M_I, M_O such that the constraints in Equations 1- 5 are satisfied.

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

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

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

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

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

The difference between NPNP-v2 and NPNP-equivalence problem are summarized as follows:

- Inputs of g can be bound to constant
- Multiple inputs of g can be bound to the same input of f.

III. Overview

Our Boolean matching engine is based on the framework of [2], as in Figure 1. Given two functions, we first construct two And-Invert Graphs (AIGs). The mapping solver will then generate possible input and output mapping. Then, the miter solver will check whether the mapping is feasible. If it is feasible, then the mapping is the matching result. If not, then the procedure will learn from this solution and strength the mapping solver. The details of the mapping solver and the miter solver will be discussed in Section IV and Section V respectively.

IV. Mapping solver

In this section, we introduce the constraints in addition to Equation 1- 3. For Section IV-A, we mention weaker lemmas about sizes of functional support of matched PI/PO from [2]. In Section IV-B, we consider the bus constraints generated from bus information of each circuit.

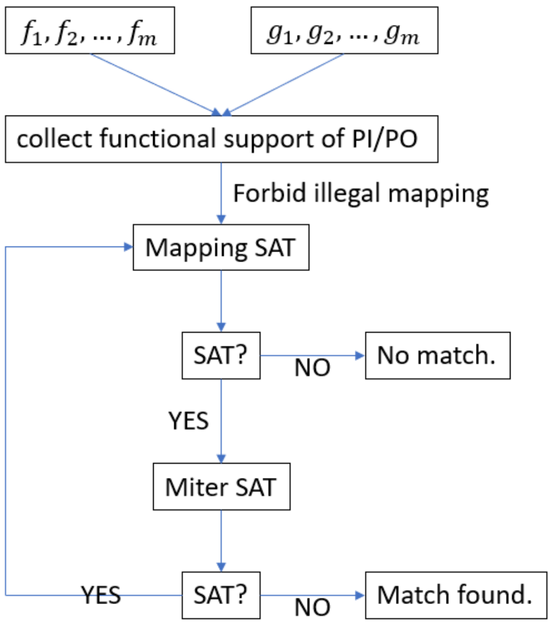

Fig. 1. Framework

A. *Functional constraints*

To introduce the functional constraints, we follow the definition in [3], and shows NPNP-v2 version lemma about the size of functional support of PI/PO in [1].

Definition 1. 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 2. 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 $f_i(x_i)$.

Based on the **Lemma 2.** in [1], two inputs (outputs) can match only if they have the same functional support size under PP-equivalence problem. Under the NPNP-v2 problem, there is a weaker lemma as follows:

Lemma 1. Let x_i and y_j be some primary input of cir0 and cir1 respectively. x_i and y_j can match if $|FuncSupp(x_i)| \leq |FuncSupp(y_j)|$.

Proof. Under NPNP-v2 problem, each primary output of cir0 maps to a primary output of cir0. Therefore, if some primary output f_i in cir0 is a functional support of x_i, then the primary output of cir1 mapped by f_i is a functional support of the primary input of cir1 mapped by x_i. If y_j is mapped to x_i, then $|FuncSupp(y_j)| \geq |FuncSupp(x_i)|$.

Lemma 2. Let f_i and g_j be some primary output of cir0 and cir1 respectively. f_i and g_j can match if $|FuncSupp(f_i)| \leq |FuncSupp(g_j)|$.

B. *Bus constraint*

Example. Suppose cir0 has two buses $W_1 = \{f_1, f_2\}$ and $W_2 = \{f_3, f_4\}$ and cir1 has $Z_1 = \{g_1, g_3\}$ and $Z_2 = \{g_2, g_4\}$. Then, we construct a bus mapping matrix

$$\begin{array}{c} \\ Z_1 \\ Z_2 \end{array} \begin{array}{c} \begin{array}{cc} W_1 & W_2 \end{array} \\ \begin{pmatrix} e_{1,1} & e_{1,1} \\ e_{2,1} & e_{2,1} \end{pmatrix} \end{array},$$

where the bus W_i is mapped to Z_j if $e_{j,i}$. As shown in Equations 6, we can restrict the the PI/PO mapping with mapped buses. For example. if $e_{1,1} = 1$, then

$$\begin{aligned} &(\neg useBus \vee \neg e_{1,1} \vee c_{1,1} \vee d_{1,1} \vee c_{1,2} \vee d_{1,2}) \\ &(\neg useBus \vee \neg e_{1,1} \vee c_{3,1} \vee d_{3,1} \vee c_{3,2} \vee d_{3,2}) \end{aligned} \tag{6}$$

When $useBus$ is set to 1, $e_{1,1} = 1$ implies g_1 and g_3 can only be mapped to f_1 and f_2.

To generate bus constraints in mapping solver, we first construct PO/PI bus mapping matrix, and restrict the PO/PI mapping as previous example.

V. Miter Solver

Our miter solver is based on the framework mentioned in [2]. The mapping solution of mapping solver is examined by solving the miter shown in Equation 7

$$\Phi = \varphi_I \wedge \varphi_O, \tag{7}$$

where

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

and

$$\varphi_O = \bigvee_{c_{i,j}+d_{i,j}=1} (f_i(X) \equiv X) \wedge (g_j(Y) \equiv g_j) \wedge (\phi_O \circ \psi_O(f_i) \equiv g_j)$$

If the solver is unsatisfiable, then a valid PO/PI mapping under NPNP-v2 is found. If satisfiable, then the solution is a counter example to the matching result from mapping solver. To prevent the counter example from happening, [3] proposes a NP3 version of the **Prop 5.** in [2]. However, there is a typo for their learned clauses, so we mention the correct learned clauses as Equation 8 and gives the proposition under NPNP-v2, which is also applicable in NP3-equivalence.

Proposition 1. Given two functions vectors $\vec{f}(\vec{x})$ and $\vec{g}(\vec{y})$ for Boolean matching under NPNP-v2, if under ϕ, ψ satisfying mapping solver there exists some $\phi_O \circ \psi_O(f_p)(\vec{u}) \neq g_q(\phi_I \circ \psi_I(\vec{v}))$, then we can prune the solution space of mapping solver by the following equation

$$\bigwedge_{p,q=1}^{n_O} (l^O_{p,q} \vee \bigvee_{i=1}^{n_I} \bigvee_{j=1}^{m_I} l^I_{i,j} \vee \bigvee_{i=1}^{n_I} c_i), \tag{8}$$

where

$$l^O_{p,q} = \begin{cases} \neg c_{q,p}, & \text{if } f_p(\vec{u}) \neq g_q(\vec{v}) \\ \neg d_{q,p}, & \text{if } f_p(\vec{u}) = g_q(\vec{v}) \end{cases},$$

$$l^I_{i,j} = \begin{cases} a_{i,j}, & \text{if } x_j \neq y_i \\ b_{i,j}, & \text{if } x_j = y_i \end{cases}, \quad c_i = \begin{cases} a_{i,m_I+1}, & \text{if } y_i = 1 \\ b_{i,m_I+1}, & \text{if } y_i = 0 \end{cases}$$

To derive more counter example, we may consider the redundant set defined in [3].

Definition 3. 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 bound to $\vec{x}$.

Example. If $\vec{x} = (0,0,1,0)$ is a counter example and $\forall x_1, x_2 . f_p(x_1, x_2, 1, 0) = 1$, then $x_1, x_2 \in R_{f_p}$.

With redundant set, we can further prune the solution space of mapping solver with Equation 9.

$$\bigwedge_{p,q=1}^{n_O} (l^O_{p,q} \vee \bigvee_{y_i \notin R_{g_q}} \bigvee_{x_i \notin R_{f_p}} l^I_{i,j} \vee \bigvee_{y_i \notin R_{g_q}} c_i). \tag{9}$$

In [3], a greedy method is proposed to find redundant set with an empty set initially. However, since we have collected functional support of each primary output, the algorithm can be sped up by setting the redundant set as the set of primary inputs not in functional support of the primary output at initial, as shown in Algorithm 1.

Algorithm 1 Get redundant set R_{f_i} of $\vec{x}$ on f_i

```
Require: a function $f_i$, an input instance $\vec{x}$, and $FuncSupp(f_i)$
1: R$_{f_i}$ ← the primary inputs of f$_i$ not in $FuncSupp(f_i)$
2: for $x \in FuncSupp(f_i)$ do
3:     add $x$ to $R_{f_i}$
4:     $\forall x_j \notin R_{f_i}, x_j$ is bound to $\vec{x}$
5:     if $\exists x_j \in R_{f_i}$ is not redundant then
6:         remove $x$ in $R_{f_i}$
7:     unbind all the inputs of $f_i$.
```

VI. Experimental Result

To evaluate the engine, the algorithms are implemented in C++. The experiments were performed on a 64-bit Linux work-station with Intel Xeon E5-2630 2.20GHz CPU and 384GB memory using the benchmark provided by ICCAD 2023 [4]. The running time limit has been set to 3600 seconds. In Table I, we compare our scores with the other ICCAD 2023 contest winners in beta test. In Table II, we compare the running time of each case with or without some constraints, and we will discuss the results in Section VII.

VII. Discussion

In this section, we discuss about our experimental result in Section VII-A. In Section VII-B, we discuss about other potential constraints about symmetry and unateness. We also conduct statistical analysis of symmetric PIs and unate PIs for each case, aims to ascertain the applicability of these signatures.

Case#	Ours	1st Place	2st Place	3st Place	4rd Place	5rd Place
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00
3	0.00	1.00	1.00	0.00	0.00	0.00
4	1.00	1.00	1.00	1.00	1.00	1.00
5	0.00	1.00	1.00	0.35	0.35	0.22
6	0.00	1.00	1.00	0.04	0.04	0.04
7	1.00	1.00	1.00	1.00	1.00	1.00
8	0.00	1.00	0.14	0.14	0.00	0.00
9	0.00	1.00	0.13	0.11	0.06	0.04
10	0.00	0.53	0.47	0.45	0.45	0.42

TABLE I
BOOLEAN MATCHING QUALITY COMPARISON WITH ICCAD 2023 CONTEST WINNERS

Case#	1	2	4	7
all constraints (s)	0.0097	5.57	0.72	1.45
w/o bus constraints (s)	0.022	9.50	0.53	3.90
w/o functional constraints (s)	0.023	80.63	1.45	47.01
w/o learned clauses (s)	13.665	>3600s	>3600s	>3600s

TABLE II
COMPARISON OF EXECUTION TIME OF EACH CONSTRAINT

A. *Comparison of constraints*

Table II demonstrates the impacts of various constraints on execution time across different cases. Since the running time of the other cases exceed 3600s, we only discussed about Case 1, 2, 4, 7. Notably, it becomes evident that functional constraints and learned clauses exert substantial influence, while bus constraints exhibit comparatively minor effects.

In particular, the "w/o learned clauses (s)" row reveals a noteworthy escalation in execution times for Case 2 and 7 when learned clauses from miter solver are omitted. This represents learned clauses from miter solver plays a crutial role in enhancing the efficiency of the Boolean matching engine. Similarly, the "w/o functional constraints (s)" row indicates a substantial increase in execution time in Case 2, 7 when functional constraints are excluded, emphasizing their crucial contribution to the efficiency.

Conversely, the "w/o bus constraints (s)" row demonstrates a relatively modest impact on execution times for Case 2 and 7 when bus constraints are disregarded. In comparison to functional constraints and learned clauses, bus constraints exert a lesser influence on the Boolean matching engine.

B. *Other potential constraints*

Within our Boolean matching engine, functional supports exclusively served as signatures to forbid certain impossible PI/PO mappings. Nevertheless, despite this approach, additional time is still necessary to ascertain correct PI mappings in certain instances. For instance, in Case 8, there exist 15 primary inputs (PIs) with a functional support size of 7 in cir0, and 18 PIs with a functional support size of 7 in cir1. These specific PIs require additional constraints to splitting them into different groups. Next, we discuss about some potential constraints for other cases, and we first give definition of symmetric PI and unate PI.

Defintion 4. Given a function $\vec{f}(X)$, x_i is a *symmetric PI* of the function $\vec{f}(X)$ if there exists some primary output f_k is symmetric on x_i and some other primary inputs x_j, i.e., $f_k(\vec{x}|_{x_i=0,x_j=1}) = f_k(\vec{x}|_{x_i=1,x_j=0})$, and $x_i \in FuncSupp(f_k)$.

Defintion 5. Given a function $\vec{f}(X)$, x_i is a *unate PI* of the function $\vec{f}(X)$ if there exists some primary output f_k is unate on x_i and $x_i \in FuncSupp(f_k)$. If x_i is not unate in f_k, then x_i is said to be binate in f_k.

Table III includes information on the number of primary inputs (#PI), the number of symmetric PIs(#sym), and the number of unate PIs (#unate) of each circuit in each case. In Case 9, the process of gathering symmetric Primary Inputs (PIs) and unate PIs is excessively time-consuming, and as a result, it has not been completed.

First, we discuss about the symmetry constraints on primary outputs. As the number of primary inputs in cir0 is equal to that in cir1, the problem is equivalent to NPNP-equivalence problem. Under this case, the mapped primary outputs must have an equal number of symmetric groups and symmetric PIs. Similarly, under NPNP-equivalence, outputs can match if they have the same number of unate and binate PIs.

For primary inputs, in [3], symmetry under NP and NP3 conditions are discussed. For NP condition, the mapped output pair (f_i, g_j) have the same size of functional support, so their primary inputs can match if their corresponding symmetric groups share the same size. Similarly, under NP condition, their primary inputs can match if they are both unate or binate.

For symmetry under NP3 condition, [3] introduce the definition of hard symmetry group of an output matching pair (f_i, g_j), and propose a theorem: given a hard symmetric group w of g_j under the output matching (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 unate under NP3 condition, the unate PI in cir1 can only be mapped to the unate PI in cir0, bound to constant or an input of cir1. Conversely, the binate PI in cir0 can only be mapped to the binate PI in cir1.

Case#	#PI		#sym		#unate	
	cir0	cir1	cir0	cir1	cir0	cir1
1	5	5	5	5	2	2
2	12	12	12	12	0	0
3	41	41	0	0	0	0
4	12	12	2	2	0	0
5	178	178	57	57	26	26
6	82	86	32	36	0	0
7	12	14	0	0	0	0
8	32	36	4	0	0	0
9	241	256	-	-	-	-
10	183	207	173	200	54	63

TABLE III
SIGNATURE OF EACH CASE

VIII. Conclusion

In this contest, we define the Boolean matching variant i.e. NPNP-V2 from NPNP. Based on the method in [3], we modify the matching constraint according to the requirement of contest and propose our own bus and functional support constraints. Furthermore, we also use learned clause form [2] and [3] to substantially speed up the solving process of minisat and successfully solved case 1, 2, 4 and 7. However, because of lack of time, we fail to add constraint for symmetry.

References

[1] H. Katebi and I. L. Markov, ”Large-scale boolean matching”, IEEE/ACM Proceedings Design Automation and Test in Eurpoe (DATE), pp. 771-776, 2010.

[2] C. F. Lai, J. Jiang and K. H. Wang, ”Boolean matching of function vectors with strengthened learning”, Proc. ICCAD, pp. 596-601, Nov. 2010.

[3] C.-W. Pui, P. Tu, H. Li, G. Chen and E. F. Y. Young, ”A two-step search engine for large scale Boolean matching under NP3 equivalence”, Proc. 23rd Asia South Pacific Design Autom. Conf. (ASP-DAC), pp. 592-598, Jan. 2018.

[4] C. -H. Chou, C. -J. J. Hsu, C. -A. R. Wu, K. -H. Tu and K. -Y. Khoo, ”Invited Paper: 2023 ICCAD CAD Contest Problem A: Multi-Bit Large-Scale Boolean Matching,” 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp. 1-4, 2023.