LSV FINAL : ICCAD Boolean Matching

I. Introduction

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.

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

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

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.

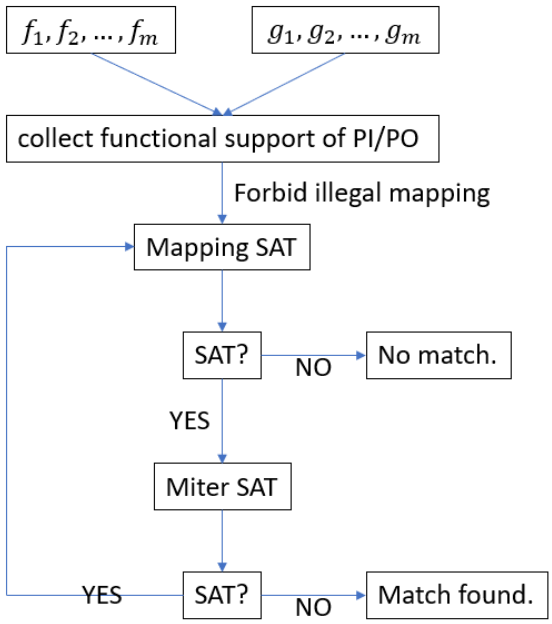

Fig. 1. Framework

A. Functional constraints

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.

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 ?????????????????? CPU and ?????GB 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 and the other ICCAD 2023 contest winners. 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.

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				
w/o functional constraint				
w/o bus constraints				
w/o				
w/o				

TABLE II
COMPARISON OF EXECUTION TIME OF EACH CONSTRAINT

VII. Discussion

VIII. Conclusion

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.

algorithm algpseudocode

Algorithm 1 Match PO with funcSupp size = 1

Require: $cir1.poNum = cir2.poNum$
0: $accum1 \leftarrow 0$
0: $accum2 \leftarrow 0$
0: $count1[cir1.piNum] \leftarrow 0$
0: $count2[cir2.piNum] \leftarrow 0$
0: $match[cir1.piNum] \leftarrow false$
for $po \in cir1$ **do**
  $count1[po.suppSize] \leftarrow count1[po.suppSize] + 1$
end for
for $po \in cir2$ **do**
  $count2[po.suppSize] \leftarrow count1[po.suppSize] + 1$
end for
for $i = 1 \rightarrow cir1.piNum$ **do**
  $accum1 \leftarrow accum1 + count1[i]$
  $accum2 \leftarrow accum2 + count2[i]$
  if $accum1 \leq accum2$ & $count1[i] == count2[i] == 1$ **then**
    $match[i] \leftarrow true$
for $po1 \in cir1$ **do**
  for $po2 \in cir2$ **do**
    $size1 \leftarrow po1, suppSize$
    $size2 \leftarrow po2, suppSize$
    if $size1 == size2 == 1$ & $match[size1]$ **then**
      add $po1.var \oplus po2.var$ to solver
=0