LSV Final Report 2018

B04901018 Sheng-Jung Yu, B04502032 Tung-Wei, Lin

***Abstract*—Boolean relations are more flexible than boolean functions because of its non-determinism. However, circuits are implementations of deterministic behaviors; therefore, the problem of determinizing boolean relations arises. Specifically, range-preserving determinization is explored in this project. We propose three methods to address this problem, and conduct experiments to compare the performances.**

***Index Terms*—Determinization of Boolean Relations, Maximal Clique, SAT problems**

I. Introduction

Boolean relations are more powerful at representing mappings than functions. For example, given a boolean mapping $\{(x_1, x_2) \in \mathbb{B}^2 \rightarrow (y_1, y_2) \in \mathbb{B}^2\}$, where $\{(0,0) \rightarrow (0,0)\}$ and $\{(0,0) \rightarrow (1,1)\}$, it is impossible to represent this one-to-many mapping using a boolean function. However, a boolean relation $\mathbf{R} = \bar{x_1}\bar{x_2}\bar{y_1}\bar{y_2} + \bar{x_1}\bar{x_2}y_1y_2$ can capture this flexibility in its entirety. In fact, boolean relations are a generalization of incompletely specified logic functions. When the output space y is restricted to $\mathbb{B}^1$, a one-to-many mapping can always be represented as don't cares.

There are mainly two ways to determinize a boolean relation, which are *range-preserving* and *deterministically reducing* [1]. On one hand, in range-preserving determinization, parametric variables are introduced in addition to the existing input variables, which is tantamount to expanding the carrier set $\mathbb{B}$. With the help of the additional variables, one could preserve all the original mappings. On the other hand, to deterministically reduce a boolean relation, one would try to choose a one-to-one mapping among the one-to-many mapping so that the final circuit implementation would have the least cost. In this project, we focus on the former.

II. Related Works

Range-preserving determinization can be useful in the context of circuit verification. To prevent the intermediate BDD from blowing up and to avoid *false negateives*(when the verification tool fails to tell the equivalences of circuits), HH Kwak *et al.* [2] propose to use K-set preserving range computation to parameterize the relations heuristically found in a circuit. By introducing new variables and quantifying out originally existent variables, they can modify the functions while preserving the output range of the original functions. In this way, the number of variables and the size of BDD can be reduced.

III. Problem Formulation

In this project, we explore the topic of ***Range-Preserving Determinization of Boolean Relations***, which is described as follows. Given a multi-output relation $\mathbf{R}(\overrightarrow{x}, \overrightarrow{y})$, where non-determinism, i.e. one-to-many mapping, exist, what is the minimum number of input variables needed to be additionally introduced in order to determinize $\mathbf{R}$.

TABLE I: **An example of a non-deterministic relation and the resultant relation after determinization.** On the left of the table is the non-deterministic relation $\mathbf{R}(x_1x_2, y_1y_2) = \bar{x_1}\bar{x_2}y_1\bar{y_2} + \bar{x_1}\bar{x_2}\bar{y_1}y_2 + x_1\bar{y_1}\bar{y_2} + x_2\bar{y_1}\bar{y_2}$. On the right, we add an additional variable x_3 to distinguish the two conflicting outputs when $(x_1, x_2) = (0, 0)$, which determinizes the relation.

x_1	x_2	y_1	y_2	x_1	x_2	$\mathbf{x_3}$	y_1	y_2
0	0	1	0	0	0	**0**	1	0
0	0	0	1	0	0	**1**	0	1
1	-	0	0	1	-	**-**	0	0
-	1	0	0	-	1	**-**	0	0

A toy example is given here to further illustrate. A boolean relation $\mathbf{R}(x_1x_2, y_1y_2) = \bar{x_1}\bar{x_2}y_1\bar{y_2} + \bar{x_1}\bar{x_2}\bar{y_1}y_2 + x_1\bar{y_1}\bar{y_2} + x_2\bar{y_1}\bar{y_2}$ in the form of a table can be found on the left in Table. I. We can see that when $(x_1, x_2) = (0, 0)$, it corresponds to two conflicting outputs, which are $(y_1, y_2) = (1, 0)$ and $(y_1, y_2) = (0, 1)$. It is shown on the right in Table. I that we distinguish the two conflicting outputs by adding an input variable $\mathbf{x_3}$, which in turn determinizes this relation $\mathbf{R}$.

This problem becomes straightforward when we expand all the cubes in $\mathbf{R}$ into minterms. After iterating through all the minterms, we can find the input with the largest number of conflicting outputs, M. Therefore, we can simply add $log_2(M)$ input variables to determinize $\mathbf{R}$. However, this approach requires that all the cubes be represented in minterms, which takes exponential time. In the following secion, we give a proof that finding the minimum number of variables in order to determinize a boolean relation is NP-Complete.

IV. Preliminaries

Before proceeding to prove the NP-Completeness of the determinization of boolean relations. We would like to discuss some simple properties, which are the stepping stones to our proof.

Lemma 1. *Given a set of cubes $\mathbf{C_n} = \{c_1, c_2, c_3, ..., c_n\}$, if the pairwise intersection of any two cubes, $c_i \cap c_j$, where $i \neq j$ and $i, j \leq n$,is not empty, then $c_1 \cap c_2 \cap c_3 \cap ... \cap c_n \neq \phi$.*

Proof. Since $c_i \cap c_j \neq \phi$, c_i and c_j should be unate in every variable. Now, say that another cube c_k also has common minterms with c_j, it means that c_k and c_j are unate in every variable as well. This in turn indicates that c_i, c_j, and c_k are all unate in every variable. If the set of cubes $\mathbf{C_n}$ is unate in every variable, we can extract at least one common minterm m from it by the following procedure.

For each variable x_i in cube $c \in \mathbf{C_n}$, if x_i is positive unate in $\mathbf{C_n}$, we set $x_i = 1$ in m; on the other hand, if x_i is negative unate in $\mathbf{C_n}$, we set $x_i = 0$ in m. Finally, if x_i appears as don't care in every cube in $\mathbf{C_n}$, we can randomly set $x_i = 0$ or $x_i = 1$ in m. ■

Lemma 2. *Maximal-Clique is an NP-Complete decision problem stated as follows. Given a graph $\mathbf{G}(V, E)$ and an integer $k > 0$, does the graph $\mathbf{G}$ conatin a maximum clique of size k?*

In [3], Karp *et al.* have given a simple outline of the proof. However, we would like to explain it in depth here.

Proof. Given an instance Φ of satisfiability problem with a set of clauses C, where the cardinality of C is m. We can construct a graph $\mathbf{G}$ for the maximal-clique problem as follows.

For each clause C_i in C, and each literal L_{ij} in clause C_i, we can build a vertex V_{ij}. An edge will be inserted between V_{ij} and V_{kh} if $i \neq k$ and $L_{ij} \neq \neg L_{kh}$. There is a clique of size m in $\mathbf{G}$ if and only if Φ is satisfiable. Because every pair of clauses are connected in $\mathbf{G}$ through non-conflicting assignment of variables($L_{ij} \neq \neg L_{kh}$). We can set all the vertices in the clique to be 1 and Φ would be satisfied. Since the transformation can be done in polynomial time($O(m \times n)$, where n is the total number of literals), maximal-clique problem is NP-Complete. ■

Theorem 1. *The determinization of a boolean relation is an NP-Complete problem stated as follows. Given a non-deterministic relation $\mathbf{R} = (\overrightarrow{x}, \overrightarrow{y})$, and an integer $k > 0$, can $\mathbf{R}$ be determinized by introducing $\lceil log_2 k \rceil$ additional variables?*

Proof. Given a graph $\mathbf{G}(V, E)$, in which a clique $\mathbf{Q} = (V_Q, E_Q)$ of size $|V_Q|$ exists, we iterate through every $v_i \in V$ and find the v_i with the maximal number M of edges $e_{ij} \in E$. This can be done in $\Theta(|V|)$. We set the input part of the relation, x, to have $|V|$ bits and the output part, y, to have $\lceil log_2 M \rceil$ bits. Then for every v_i in V, we assign an mapping of $x \rightarrow y$ to it, where x can possess dont care bits while y is fully specified.

Initially we set all bits in both input and output parts to have dont care as initial state. We iterate through $vi \in V$, and for every v_j that has an edge e_{ij} connecting it to v_i , we assign the output parts y_i and y_j to be different (we will not run out of distinct y because we allocated M bits in the beginning) and for every v_k that has no edge connecting it to v_i, we assign the i^{th} of their input part x_k to be 1. Last, we assign the i^{th} bit of x_i to be 0.

In this way, vertices that are connected will have conflicting outputs and are unate in the input part. Additionally, the input part of the vertices that have no edges in between will differ at least in one bit. From Lemma. 1, we can see that the vertices V_Q in the maximal-clique Q are unate in the input part while having different outputs. This in turn indicates that there is at least one common minterm within V_Q.

Hence, we would need $\lceil log_2(|V_Q|) \rceil$ variables in order to determinize the constructed relation. This procedure can be done in polynomial time ($O(E \times V)$); therefore, the determinization of a boolean relation is NP-Complete. ■

Theorem 2. *A Max-SAT problem is an NP-Complete decision problem stated as follows. Given a conjunctive normal form (CNF) formula $\mathbf{F}$ and an integer $k > 0$, is there a truth assignment that can satisfy at least k clauses simultaneously in $\mathbf{F}$?*

It is intuitively easy to see that a Max-SAT problem falls under the NP-Complete category since it is a generalization of the SAT problem. However, we still try to provide a formal proof here.

Proof. Given an instance Φ of satisfiability problem with a set of clauses C, where the cardinality of C is m. We can construct a Max-SAT instance Ψ in the following way.

For each clause c_i in C, in which x_i literals exist, we introduce a dummy literal α_i and construct a c_i' for Ψ. Without loss of generality, we assume $c_i = (\ell_1 \vee \ell_2 \vee ... \vee \ell_x)$. Then, we construct

$$(\ell_1)(\ell_2)...(\ell_x)(\alpha_i) \quad (1)$$
$$(\neg\ell_1 \vee \neg\ell_2 \vee ... \vee \neg\ell_{x-1}) \quad (2)$$
$$(\neg\ell_1 \vee \neg\ell_2 \vee ... \vee \neg\ell_{x-2} \vee \neg\ell_x) \quad (3)$$
$$\dots \quad (4)$$
$$(\neg\ell_2 \vee \neg\ell_3 \vee ... \vee \neg\ell_x) \quad (5)$$
$$(\ell_1 \vee \neg\alpha_i)(\ell_2 \vee \neg\alpha_i)...(\ell_x \vee \neg\alpha_i), \quad (6)$$

There are $x_i + 1 + C^{x_i}_{x_i-1} + x_i = 3x_i + 1$ clauses in c_i'. We claim that if c_i is to be satisfied in Φ, at least $2x_i + 1$ clauses in c_i' should be satisfied as well. This is because the only way to falsify c_i is to assign ℓ_i to "False", $\forall i \leq x_i$. This assignment will satisfy clauses from (2) to (5), which are x clauses. If α_i is assigned "False", then another x_i clauses in (6) will also be satisfied. On the other hand, if α_i is assigned "True", only 1 clause, which is the last clause, (α_i), in (1) will be satisfied. Due to the nature of Max-SAT, it is preferred that α_i is assigned to "False". Hence, the number of satisfied clauses in c_i' is $2x$ when c_i is falsified. In all other cases, where there is at least one literal in c_i is assigned "True", the number of clauses satisfied in c_i' will be greater than $2x$. For instance, say l_i is the only literal assigned "True" in c_i, and that α_i is assigned "False". Aside from the $2x$ clauses from (2) to (6), one clause in (1) is also satisfied, namely (ℓ_i). While we don't analyze every case here, the reader is encouraged to go through each possible case to verify.

From the above, we can set the k for Ψ to be $\sum_{i=1}^{m}(2x_i+1)$. If Max-SAT fails to find an assignment to satisfy k clauses, Φ is UNSAT; while when such an assignment exists, Φ should be SAT. Since the translation of SAT to Max-SAT can be done in polynomial time, Max-SAT is NP-Complete. ■

V. Proposed Method

We have shown in Section. IV the complexity of determinizing a boolean relation. In the following we propose three methods to solve this problem.

A. Conversion to Undirected Graph

It can be shown that a relation $\mathbf{R}$ can be converted into a graph $\mathbf{G}(V, E)$ by the following procedure. Given a relation

TABLE II: **An example of expanding a relation.** On the left of the table is the original relation; while on the right, is the relation after expanding on r_2 and r_3.

	x_1	x_2	y_1	y_2		x_1	x_2	y_1	y_2
r_1	-	-	0	1	r'_1	-	-	0	1
r_2	-	1	1	-	r'_2	-	1	1	0
					r'_3	-	1	1	1
r_3	0	-	-	1	r'_4	0	-	0	1
					r'_5	0	-	1	1

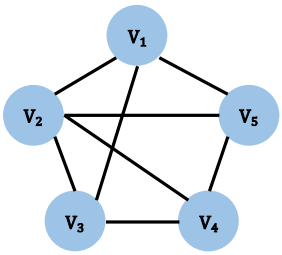

Fig. 1: **The resultant graph of the example relation in Table. II.**

$\mathbf{R}(\overrightarrow{x}, \overrightarrow{y})$, we first expand the output part $\overrightarrow{y}$ into minterms while keeping the input part $\overrightarrow{x}$ as it is. Hence, we get a relation in the form of a table with n rows. We present an example in Table. II. Originally, the relation has only three rows while r_2 and r_3 have don't care bits in the output. After expansion, we get $n = 5$ on the right in Table. II .

Then for each row r'_i, we construct a vertex v_i, which constitutes the set of vertices V. As for the construction of edges, we compare each pair of rows r'_i and r'_j, where $1 \leq i, j \leq n$ and $i \neq j$. If r'_i and r'_j are unate and the respective output parts $\overrightarrow{y_i}$ and $\overrightarrow{y_j}$ are different, an edge between v_i and v_j is inserted. The resultant graph from the example in Table. II is shown in Fig. 1.

From Lemma. 1, we can see that all the vertices in the maximum-clique Q in the constructed graph $\mathbf{G}$ have common minterms. By adding $\lceil log_2(|Q|) \rceil$ variables, we are able to determinize $\mathbf{R}$. With the obtainment of a graph, it is possible to find all maximal-cliques by using the Bron-Kerbosch algorithm [4], which does recursive backtrack on every vertex until a maximal-clique is confirmed. Then, for all the maximal-cliques found, we choose the largest, which is the maximum-clique in $\mathbf{G}$.

B. Conversion to Max-SAT Problem

Besides converting the relation in table format into an undirected graph, we can also convert it into a CNF formula and feed the formula into any Max-SAT solver, e.g. QMaxSAT [5]. In [6], a procedure of converting a max-clique problem to Max-SAT formula is given. Here, we provide a similar procedure to directly convert from the relation.

Given a relation $\mathbf{R}$ with n rows, we allocate $n + 1$ literals, including n literals, where each literal ℓ_i corresponds to each row r_i, and a dummy literal z. For each row r_i in the table, we construct two clauses $(\ell_i \vee z)(\ell_i \vee \neg z)$. Then, we iterate through r_j, where $i < j \leq n$, and construct a clause $(\neg \ell_i \vee \neg \ell_j)$ if the row r_j has no conflict with r_i, meaning that either the input parts of r_i and r_j are binate or that the output parts are identical. In conclusion, the CNF formula constructed would be

$$F = \prod_{\substack{1 \leq i \leq n \\ i < u, v \leq n}} (\ell_i \vee z)(\ell_i \vee \neg z)\ (\neg \ell_i \vee \neg \ell_u)...(\neg \ell_i \vee \neg \ell_v), \quad (7)$$

in which rows r_u and r_v have no conflict with r_i.

After feeding F into a Max-SAT solver, the solution should contain an assignment of the literals that makes the maximal number of clauses satisfied. Taking $\lceil log_2 \rceil$ of the number of literals(excepting z), Q, that are assigned "True" should be the minimum number of variables introduced to determinize $\mathbf{R}$. A brief explanation of the rationale behind this is provided below while a more comprehensive study can be found in [6]. The more literals that are assigned "True" in F, the more clauses in the form $(\ell_i \vee z)(\ell_i \vee \neg z)$ will be satisfied. Additionally, clauses in the form $(\neg \ell_i \vee \neg \ell_u)$ encourage rows that are in conflict to be assigned "True".

C. Unate Splitting with Branch and Bound

The algorithms proposed in Sections V-A and V-B require complete expansion of all cubes of y. However, complete expansion results in exponential number of rows, and thus the resultant graph or CNF formula is exponential in their size. As a result, the input size of the NP-hard problem becomes exponential, which prohibitively increases the time complexity of the problem. To address this issue, in this section we propose a expansion-free algorithm based on the concept in the previous sections. The flow of the algorithm is summarized in Algorithm 1.

To explain the algorithm, we define a graph $G(V, E)$ similar to the graph in V-A. Every row r_i in relation is represented by a vertex v_i in V, and a edge $e = (v_i, v_j)$ exists if rows r_i, r_j have common cubes, as shown in the Figure 2.

Algorithm 1 UnateFinding

Require: The Bounding size y_c, Cubes of a relation $\mathbf{R}$, C_R.
Ensure: The maximum number of y relating to a single mintern of x.
1: y'_c = *EstimateMintern*(C_R) // *upper bound estimation of the minterns*
2: **if** $y'_c \leq y_c$ **then**
3: **return** y_c
4: **if** *isUnate*(C_R) **then**
5: **return** *CountMintern*(C_R)
6: (C_0, C_1) = *Splitting*(C_R) // *cofactoring a non-unate variable.*
7: y_c = *UnateFinding*(y_c,C_0)
8: y_c = *UnateFinding*(y_c,C_1)
9: **return** y_c

Since the number of additional variables needed corresponds to largest number of y minterns contained by a single x mintern, from the following Lemmas 3 and 4, the number of y minterns in the clique having most number of y minterns, y_c, is the solution.

Lemma 3. *Given a set of cubes* $\mathbf{C_n} = \{c_1, c_2, c_3, ..., c_n\}$*, the set of cubes is unate iff the set of cubes has a common cube* $c_1 \cap c_2 \cap c_3 \cap ... \cap c_n \neq \phi$.

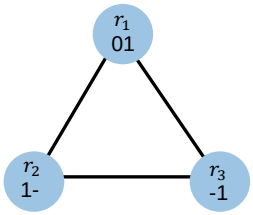

Fig. 2: **The example of the graph constructed by relation in Table II.**

Proof. 'Only if' part: The proof of the 'only if' part of this lemma similar to Proof IV. If a set of cubes is unate, then for every literal l_i, we choose l_i if the cubes are positive unate in l_i and choose l_i' if the cubes are negative unate in l_i. The resulting cubes must be a common cube of all cubes since there are no conflict, i.e. the literal is positive in one cube and negative in another cube, in all literals against any cube in $\mathbf{C_n}$, which is the property of unate.

'If' part: If $c_{common} = c_1 \cap c_2 \cap c_3 \cap ... \cap c_n$ is not empty, Then in c_{common}, for each literal l_i, there must be no conflict between the common cube and any cube in $\mathbf{C_n}$. Therefore there must be no two cubes in $\mathbf{C_n}$ having conflict literals, which means every literal is unate in $\mathbf{C_n}$. ■

Lemma 4. *all sets unate cubes in the relation is a clique in the graph $G(V, E)$ introduced in Section V-A*

Proof. From Lemma 3, there is a common cube in $\mathbf{C_n}$, any two of the cubes must have a common cube and therefore is connected by an edge, which forms a clique. ■

Instead of searching all cliques in the graph, since the number of minterns in y is monotonic increasing, only maximal cliques are candidates of having most number of minterns. Therefore, we can apply unate splitting, with cofactor literals chosen in branchs being the non-unate literals, to find all maximal cliques.

Theorem 3. *all maximal clique in the graph $G(V, E)$ is a unate leaf if the splitting variable is chosen from non-unate literal.*

Proof. After splitting a non-unate literal, the rows without conflict at the literal are kept in the same branch and therefore no cliques are break. Therefore the maximal clique is kept until the unate leaf and no splitting can be made. ■

Given the above property, we can splitting the row into unate leaf to find all maximal clique and compute the number of unique y minterns contained in the corresponding rows to find y_c. Although computing the number of all unique y minterns is still exponential worst-time complexity, the y minterns can be expressed in a more compact form without expansion which therefore reduces problem size. In addition, since the upper bound of the number of y minterns can be computed in $O(n)$ by the summation of the cubes size in every row. We combined the algorithm with branch and bound to further reduce the computation time on some unate leaves that have less minterns of y by bounding the branch with the current most number of y minterns. After we find y_c, the required number of input variable are $\lceil log_2(y_c) \rceil$.

VI. Experimental Results

We implemented the proposed methods in the c++ programming language. All experiments were conducted on a Intel Xeon 3.5GHz Linux Workstation with 76GB memory and the testcases in the experiments were extracted from the ISCAS benchmarks. The maximum clique problem was solved by [7] and the Max-SAT problem was solved by QMaxSAT [5]. To compare the efficiency of the methods, we ran the methods on the same testcases and record their execution time. From the experimental results, the unate splitting method completes all testcases in acceptable time, while the MaxSAT method cannot finish most of the testcases and graph method fails at large testcases. The execution time of unate splitting method is minimal among 3 method in all testcases, as shown in the Table III. The reason why unate leaf method outperformed max clique and Max-SAT methods is that we save the time spent on expansion of y minterns. The expansion of y mintern and creation of edges alone even took longer time than the whole unate leaf method.

benchmark					Graph	MaxSAT	Unate
Name	NX	NY	NM	NV	Time	Time	Time
c17_po1	2	2	3	2	1.40E-4	7.15E-3	2.63E-5
c432_po0	12	6	64	6	7.95E-2	–	1.59E-4
c432_po1	12	7	511	9	–	–	9.43E+0
3540n728	18	9	240	8	1.40E+2	–	4.30E+0
499n177	9	5	16	4	2.34E+0	–	3.16E-2
880n316	9	5	92	7	1.11E-1	–	1.38E-2
880n359	21	11	2048	12	–	–	3.61E+0

TABLE III: The number of x variables(NX), the number of y variables(NY), max number of y minterns to the same x (NM), the number of required number of additional variables(NV) and the comparisons of CPU times(secs), the notation – means that the tool crashes or could not finish in 1000 secs.

VII. Conclusion

In this project we study the boolean relation determinization problem and propose 3 methods to find the required number of additional variables. The proposed methods are based on undirected graph, Max-SAT problems and unate splitting respectively. Experimental result shows that the unate leaf method is able to efficiently handle larger problems while the other two methods are not.

References

[1] J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung, "Interpolating functions from large boolean relations," in *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*. IEEE, 2009, pp. 779–784.

[2] H. H. Kwak, I.-H. Moon, J. H. Kukula, and T. R. Shiple, "Combinational equivalence checking through function transformation," in *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*. ACM, 2002, pp. 526–533.

[3] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*. Springer, 1972, pp. 85–103.

[4] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[5] H. F. Miyuki Koshimura, Tong Zhang and R. Hasegawa., "Qmaxsat: A partial max-sat solver." https://sites.google.com/site/qmaxsat/, 2012.

[6] "A reduction of clique to max 2-sat," http://www.amotlpaa.org/math/max2sat.pdf.

[7] J. Konc and D. Janezic., "An improved branch and bound algorithm for the maximum clique problem." *Communications in Mathematical and in Computer Chemistry*, vol. 58, pp. 569–590, 2007.