Dependent Latch Identification in the Reachable State Space

Chen-Hsuan Lin Chun-Yao Wang
Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
adonis@nthucad.cs.nthu.edu.tw wcyao@cs.nthu.edu.tw

***Abstract*— The large number of latches in current designs increase the complexity of formal verification and logic synthesis, since the growth of latch number leads the state space to explode exponentially. One solution to this problem is to find the functional dependencies among these latches. Then, these latches can be identified as dependent latches or essential latches, where the state space can be constructed using only the essential latches. This paper proposes an approach to find the functional dependencies among latches in a sequential circuit by using SAT solvers with the Craig interpolation theorem. In addition, the proposed approach detects sequential functional dependencies existing in the reachable state space only. Experimental results show that our approach could deal with large sequential circuits with up to 1.5K latches in a reasonable time and simultaneously identify the combinational and sequential dependent latches.**

I. Introduction

In the formal verification of sequential systems, a main task is to test the equivalence of two given sequential circuits, also called sequential equivalence checking (SEC). Examining two sequential circuits S_1 and S_2 for equivalence can be reduced to a reachability analysis by building a product machine of the two sequential circuits (called a miter in [4]). Since the state space of a product machine is the Cartesian product of the state space of S_1 and S_2, and it grows exponentially with the increase of the number of latches in it, the corresponding reachability analysis of the machine will become more difficult. Also, the performance of SEC strongly depends on the reachability analysis [5] of the product machine. In addition to SEC, many other tasks for sequential circuits, such as property checking or model checking [19], can be reduced to a reachability analysis as well. Hence, reachability analysis plays an important role in the formal verification of sequential systems and its efficiency would influence the performance of formal verification for sequential circuits dramatically.

There are many approaches, such as early quantification [12], approximative state space traversal [7][9][20], and functional dependencies [10][24], to improve the efficiency of reachability analysis. Functional dependencies, which this work focusing on, is to identify dependent latches in a machine. Since some latches in a system might functionally depend on the other latches, it is possible that not all latches in a system have to be considered during the reachability analysis. Therefore, the identification of dependent latches, that functionally depend on the other latches within the system, plays an important role in reducing the state space of a product machine. It also helps to minimize the number of latches in a sequential circuit.

To identify the dependent latches in a sequential circuit, the functional dependencies between these latches must be discovered. The functional dependencies are the relationships between these latches, and their corresponding Boolean functions are called *dependency functions*. With information obtained about dependency function, dependent latches can be replaced by other latches. Hence, a sequential circuit might be optimized by removing these dependent latches without changing the circuit's functionality [16]. In addition, disregarding dependent latches can avoid the explosive expansion of Binary Decision Diagram (BDD) [2] size during the BDD-based reachability analysis [10].

This work was supported in part by the National Science Council of R.O.C. under Grant NSC 97-2220-E-007-042 and NSC 97-2220-E-007-034.

Equivalence relation ($F_i \equiv F_j$) and opposition relation ($F_i = \neg F_j$) are examples of some typical functional dependencies. [11] and [24] have proposed algorithms to identify them. However, there may exist many other relations among these latches. For instance, AND relation ($F_i = F_j \bullet F_k$) is another relation. An approximate approach in [13] directly extracts dependency functions from the latches' transition functions by using BDDs. But the scalability of the approach is still restricted to the BDD size. As a result, this approach might not efficiently identify all functional dependencies among latches. Therefore, another approach [15] detects the functional dependency by using a SAT solver with the Craig interpolation theorem [6]. With the great recent advances on SAT solvers [8][17], this approach can identify more functional dependencies among latches.

Although [15] exploited a SAT solver to find more functional dependencies among latches, the identified dependency functions may not be precise enough because of using maximal input support candidates. Furthermore, it only reports which latches have a dependency function, but does not explicitly indicate the dependent latches. Thus the identified results cannot be directly used for further applications. On the other hand, since [15] extracts the functional dependency from the latches' transition functions, its functional dependency is a combinational functional dependency, which holds in the whole state space. As for the sequential functional dependency, which holds only in the reachable state space, [15] cannot explore it. The sequential functional dependency is capable of identifying additional dependent latches in a circuit after a specific timeframe.

The contributions of this paper are as follows: **(I)** An ordered destroyed cost heuristic is derived to minimize the input support candidates such that as many dependent latches are identified as possible. The corresponding dependent functions among the latches are derived as well. **(II)** An efficient method for discovering sequential functional dependencies among latches by exploiting the incremental SAT technique and the early detection of dependent latches are proposed. As a result, more dependent latches could be identified at each timeframe.

II. Preliminaries

A. *Functional dependency of latches*

Definition 1: A latch's *transition function* is a Boolean function $f(X)$ whose input domain consists of the primary inputs (PIs) and pseudo primary inputs (PPIs). The PPI is the output signal of a latch. This determines the next state value of this latch.

Definition 2: Given a latch's transition function $r(X) : B^m \rightarrow B$, and a vector of other latches' transition functions $S = \langle s_1(X), \ldots, s_n(X)\rangle$, where $s_i(X) : B^m \rightarrow B$ for $i = 1, \ldots, n$ over the same input domain $\{X \in B^m | X = \langle x_1, \ldots, x_m\rangle\}$. The total number of PIs and PPIs (latches) is m. r functionally depends on S if there is a Boolean function $d : B^n \rightarrow B$, called the *dependency function*, such that $r(X) = d(s_1(X), \ldots, s_n(X))$. The latch r is called a *replaced latch*, and the latches in vector S are called *substituted latches*.

Note that the replaced latch might depend only on a subset of the substituted latch set $\{s_1(X), \ldots, s_n(X)\}$ when the dependency function is written as $r(X) = d(s_1(X), \ldots, s_n(X))$.

978-1-4244-2749-9/09/$25.00 ©2009 IEEE

B. Existence of functional dependency

A necessary and sufficient condition to examining the existence of functional dependency among latches is described as follows.

Theorem 1 [13]: Given the transition function of a replaced latch r and the transition functions of substituted latches S, let $d^0 = \{Y \in B^n | Y = S(X) \text{ and } r(X) = 0, X \in B^m\}$ and $d^1 = \{Y \in B^n | Y = S(X) \text{ and } r(X) = 1, X \in B^m\}$. The dependency function d exists if and only if $d^0 \cap d^1$ is empty. Therefore, d^0 , d^1 , and $B^n \setminus (d^0 \cup d^1)$ could be considered as the off-set, on-set, and don't care-set of d, respectively. In brief, when $d^0 \cap d^1 = \phi$, $r(X) = d(S(X))$ is always true for all input combinations of X.

Theorem 1 can be used to examine whether the dependency function d exists or not.

C. Exploration of functional dependency by SAT solvers

*1) **Identification model**:* [15] defines a general model, as seen in Fig. 1, to establish whether the dependency function exists by using a SAT solver. This method will extract the combinational part of a sequential circuit with l latches. This combinational part includes each latch's transition function with the PIs, PPIs, and pseudo primary outputs (PPOs), the input signal of latches, as the signals (the primary outputs will be ignored in this work). $x_i, i = 1, \ldots, m$ is one of PIs or PPIs and $y_j, j = 1, \ldots, l$ is one of PPO functions. Each y_j represents the next state transition function of latch j, so this latch's next state value = $y_j(x_1, \ldots, x_m)$. The combinational part of the circuit is instantiated into two copies called $Comb_{on}$ and $Comb_{off}$ to form the circuit part of the model. For every variable v in $Comb_{on}$, there is starred counterpart v^* in $Comb_{off}$. The constraint part of the model selects $(n+1)$ out of l latches which are considered in the circuit, $(n+1) \leq l$. With y_j and ${y_j}^*, j = 0, \ldots, n$ in the constraint part, $y_0 = 1$ would be set as the constraint in $Comb_{on}$ and $y_0^* = 0$ in $Comb_{off}$. This action results in the domain of X (resp. X^*) restricted to the sub-domain leading $y_0 = 1$ (resp. $y_0^* = 0$). Similarly, the domain of $\langle y_1, \ldots, y_n \rangle$ (resp. $\langle y_1^*, \ldots, y_n^* \rangle$) would be restricted to the sub-domain based on the constrained domain of X (resp. X^*).

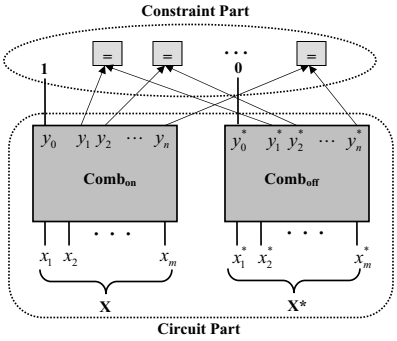

Fig. 1. The identification model for SAT solvers [15].

*2) **Model operation**:* To check whether a dependency function d exists among a replaced latch r and the substituted latches $\langle s_1, \ldots, s_n \rangle$, y_0 and $\langle y_1, \ldots, y_n \rangle$ can be regarded in this model as the replaced latch r and the substituted latches $\langle s_1, \ldots, s_n \rangle$, respectively. Then, the constrained sub-domain of $\langle y_1, \ldots, y_n \rangle$ can be taken as the on-set d^1 and the constrained sub-domain of $\langle y_1^*, \ldots, y_n^* \rangle$ as the off-set d^0. Therefore, if the sub-domain of $\langle y_1, \ldots, y_n \rangle$ and the sub-domain of $\langle y_1^*, \ldots, y_n^* \rangle$ overlap, it indicates that $d^0 \cap d^1$ is not empty and the dependency function d does not exist according to Theorem 1.

*3) **Model transformation to a SAT problem**:* The circuit part $Comb_{on}$ and $Comb_{off}$ of the model in Fig. 1 can be converted to conjunctive normal forms (CNFs) C_{on} and C_{off}, respectively [1]. Similarly, the constraint part can be converted to a CNF : $y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \ldots \wedge (y_n \equiv y_n^*)$. Therefore, the complete CNF of the identification model is

$$C_{model} = C_{on} \wedge C_{off} \wedge y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \ldots \wedge (y_n \equiv y_n^*),$$

where $(y_j \equiv y_j^*)$ stands for $(y_j \vee \neg y_j^*) \wedge (\neg y_j \vee y_j^*)$.

Finally, SAT solvers can be used to examine whether C_{model} is satisfiable or unsatisfiable.

Satisfiable: There exists one assignment of $\langle y_1, \ldots, y_n \rangle$ and $\langle y_1^*, \ldots, y_n^* \rangle$ such that $(y_0 = 1)$ and $(y_0^* = 0)$ are both true. That is, $d^0 \cap d^1$ is not empty and the dependency function d does not exist.

Unsatisfiable: No assignment of $\langle y_1, \ldots, y_n \rangle$ and $\langle y_1^*, \ldots, y_n^* \rangle$ can be found, so that $d^0 \cap d^1$ is empty and the dependency function d exists.

The summaries above are based on the following theorem in [15].

Theorem 2 [15]: Given the transition function $r(X)$ of a replaced latch and the transition functions $\langle s_1(X), \ldots, s_n(X) \rangle$ of substituted latches, a dependency function d exists between the replaced latch and substituted latches if, and only if, the corresponding C_{model} is unsatisfiable.

D. Determination of dependency function by Craig Interpolation

Theorem 3 [6]: (Craig Interpolation Theorem) In two CNFs C_a and C_b with the same common input variables $\langle v_0, \ldots, v_n \rangle$, if $C_a \wedge C_b$ is unsatisfiable, there exists a Boolean formula I only referring to the common input variables $\langle v_0, \ldots, v_n \rangle$ with the property that $C_a \Rightarrow I$ and $I \Rightarrow \neg C_b$.

This Boolean formula I is called the *interpolant* of C_a and C_b. The interpolant can be constructed in linear time from the refutation proof [14][18][21], and current SAT solvers [8][17] can easily produce it from an unsatisfiable problem.

Theorem 4 [15]: For the CNF $C_{model} = C_{on} \wedge C_{off} \wedge y_0 \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \ldots \wedge (y_n \equiv y_n^*)$, if it is unsatisfiable, we can partition C_{model} into C_a and C_b (i.e., $C_{model} = C_a \wedge C_b$) with $C_a = C_{on} \wedge y_0$ and $C_b = C_{off} \wedge \neg y_0^* \wedge (y_1 \equiv y_1^*) \wedge \ldots \wedge (y_n \equiv y_n^*)$. The common input variables of C_a and C_b are $Y = \langle y_1, \ldots, y_n \rangle$. Because the C_{model} is unsatisfiable, there exists an interpolant formula I only referring to the common variables Y, and $I(Y)$ can be taken as the dependency function d.

The detailed proof of Theorem 4 in [15] shows that $I(Y)$ must be an over-approximation of $d^1(Y)$ and must be disjoint from $d^0(Y)$. Therefore, $I(Y)$ is the valid dependency function, such that $r = I(s_1, \ldots, s_n)$ is always true.

III. The Limitations of the State of the Art

A. The difference between functional dependencies and dependent latches

The dependency functions among latches can be explored by previous work [15], but it still has a gap to the dependent latch identification. For instance, if three latches A, B, and C in a sequential circuit have circular functional dependencies: $A = f_A(B, C)$, $B = f_B(A, C)$, $C = f_C(A, B)$. The previous method could only report that three functional dependencies exist but cannot identify which latches are dependent.

B. The effect of input support candidate selection

To identify all functional dependencies in a sequential circuit, the previous work would take all the other latches as the input support candidates, named maximal input support candidates in this work, to explore the dependency function of a latch. But with maximal input support candidates, the dependency functions obtained in the previous work may contain some redundant input supports and rely on dependent latches, and these results do not benefit the dependent latch maximization. Two simple examples, as shown in Fig. 2(a) and Fig. 2(b), are used to demonstrate the effects of input support candidate selection.

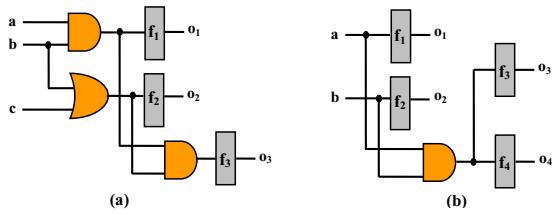

Fig. 2. Two examples demonstrating the effect of maximal input support candidates.

*1) **Dependency functions containing redundant input supports**:* Fig. 2(a) is a sequential circuit with three latches (f_1, f_2, f_3) and three PIs (a, b, c). As the previous work examines whether f_3's functional dependency exists, it would take $\{f_1, f_2\}$, the maximal input support candidates, as its input support candidates. With the proposed identification model, it sets n to 2 in the constraint part of the model, and regards y_0 and $\langle y_1, y_2\rangle$ in the model as the replaced latch f_3 and the substituted latches $\langle f_1, f_2\rangle$, respectively. As a result, it will get a dependency function $f_3 = f_1 \bullet f_2$ as can be observed from Fig. 2(a). However, if f_3's input support candidates are restricted to $\{f_1\}$, a more simplified, but not intuited dependency function $f_3 = f_1$ would be found. This is because $f_3 = f_1 \bullet f_2 = (ab) \bullet (b + c) = ab + abc = ab = f_1$. In this work, we will remove as many redundant variables from the input support candidate set as possible, to obtain more simplified dependency functions.

*2) **Dependency functions relying on dependent latches**:* Fig. 2(b) is a sequential circuit with four latches (f_1, f_2, f_3, f_4) and two PIs (a, b). As the previous work examines whether f_3's functional dependency exists, it would take $\{f_1, f_2, f_4\}$, the maximal input support candidates, as its input support candidates. As a result, it would get a dependency function $f_3 = f_4$. Dealing in the same way with f_1, f_2 and f_4, it will also get $f_4 = f_3$, and will regard f_1 and f_2 as independent latches. Although the dependency functions of f_3 and f_4 do not have redundant input supports, that is, they are simplified dependency functions, these results will significantly influence the dependent latch identification. According to the dependency functions of f_3 and f_4 explored by the previous work, only one of them can be identified as a dependent latch, so the other one should be an essential latch. In this work, however, the input support candidates of f_3 and f_4 will be restricted to $\{f_1, f_2\}$, and their corresponding dependency functions $f_3 = f_1 \bullet f_2$ and $f_4 = f_1 \bullet f_2$ will be obtained. According to the dependency functions, f_3 and f_4 can be simultaneously identified as dependent latches.

IV. The Ordered Destroyed Cost Heuristic

In this section, an example in Fig. 3 is used to demonstrate the heuristic for the dependent latch identification in sequential circuits. At first, the identification model is used to determine which latch's dependency function exists with maximal input support candidates, that is, by considering all other latches as input support candidates. But its dependency function will not currently be derived due to inaccuracy. Thus, two sets of latch $P : \{L_1, L_2, L_3, L_9, L_{10}\}$ and $I : \{L_4, \ldots, L_8\}$ are distinguished, where P is the set of possibly dependent latches in which functional dependencies exist, and I is the set of independent latches.

A. *Refinement of the set P by the set I*

Latches in I do not have any functional dependency, hence, they cannot be identified as dependent latches. That is, they are the essential latches of the circuit and can be used as the substituted latches to replace the latches in P. We can determine whether the latches in $P : \{L_1, L_2, L_3, L_9, L_{10}\}$ depend solely on the latches in $I : \{L_4, \ldots, L_8\}$. In addition, constant latches in a circuit will also be identified in this step since constant latches can be presented as

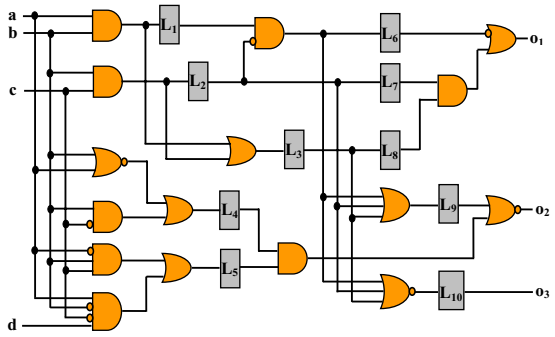

Fig. 3. A demonstrating example.

having functional dependencies over any input support candidates. For example, to examine whether L_1 depends only on the set I or is a constant latch using the identification model, the n is set to 5 in the constraint part of the model, and y_0 and $\langle y_1, \ldots, y_5\rangle$ are regarded as the replaced latch L_1 and the substituted latches $\langle L_4, \ldots, L_8\rangle$, respectively. The other latches in P can also be examined in the same manner. In Fig. 3 it can be found that L_9 and L_{10} are the latches dependent on I, and their input support candidate sets are refined from $\{L_1, \ldots, L_8, L_{10}\}$ and $\{L_1, \ldots, L_9\}$ to $\{L_4, \ldots, L_8\}$, and no constant latch exists in this example. As a result, L_9 and L_{10} are put with set R, which collects the dependent latches depending on I and the constant latches. Then, set P is updated from $\{L_1, L_2, L_3, L_9, L_{10}\}$ to $\{L_1, L_2, L_3\}$.

B. *Refinement of the set P by the ordered destroyed cost*

After the refinement by I, the latches in P either depend on some of the latches in P itself or some of the latches in P and I. That is, each latch L_i in P whose dependency function exists with input support candidates $P \backslash \{L_i\}$ or $(P \backslash \{L_i\}) \cup I$. Next, P is divided into two disjointed sets P_R and P_I, where the latches in P_R are dependent on $P_I \cup I$. The goal is to maximize the P_R set and minimize the P_I set. This is because the more latches in P_R, the more latches that can be identified as dependent latches.

An ordered destroyed cost is proposed as the guide for the heuristic to move latches from P to P_I. The same example is used to demonstrate how to evaluate and use the destroyed cost. The physical meaning of the destroyed cost will be discussed at the end of this section.

To evaluate the destroyed cost of L_3, denoted as dc_3, L_3 is removed from the input support candidates of latches L_1 and L_2, and the identification model is used to examine how many functional dependencies would not exist without L_3. That is, the number of L_j's functional dependencies with candidates $(P \setminus \{L_3, L_j\}) \cup I$ that will be destroyed by removing L_3 are counted. In this example, both L_1's functional dependency with input support candidates $\{L_2, L_4, \ldots, L_8\}$ and L_2's functional dependency with input support candidates $\{L_1, L_4, \ldots, L_8\}$ would be destroyed without L_3, according to the identification model. This results in the destroyed cost of L_3=2.

After all calculations, we obtain the destroyed cost dc_1=1, dc_2=1, and dc_3=2. According to the descending order of the destroyed cost, L_i with the highest dc_i is removed to P_I and it is checked whether the remaining latches in P depend solely on the updated $P_I \cup I$. If a latch in P satisfies this dependency condition, it can be moved to P_R. If P is not empty, L_i can be further moved with the next highest dc_i to P_I and the operations mentioned above can be iterated.

For this example, L_3 with dc_3=2 is first moved to P_I, and the remaining latches $\{L_1, L_2\}$ in P are checked to see if they functionally depend on the updated $P_I \cup I$ (updated $P_I : \{L_3\}$) using the identification model. In this iteration, $\{L_1, L_2\}$ have dependency functions with input support candidates $P_I \cup I$. Thus, they are moved

to P_R. Finally, P is empty, and the heuristic has partitioned P into $P_I : \{L_3\}$ and $P_R : \{L_1, L_2\}$. All latches in P_R functionally depend on $P_I \cup I$. The pseudo code of evaluating the destroyed cost of latches in P is shown in Fig. 4.

```
EvaluateDestroyedCost( )
  Input : set P, set I
  Output : dc_i of each L_i in P
    for each latch L_i in P
      for all other latches L_j (j ≠ i) in P{
        Check existence of FD_j with input candidates
          (P \ {L_i, L_j}) ∪ I ;
        if (remove L_i destroys FD_j)
              dc_i++ ;
      }
    return dc_i of each L_i in P ;
```

* FD_j is the functional dependency of latch L_j

Fig. 4. The pseudo code of EvaluateDestroyedCost function.

The results of this example using the proposed heuristic are shown in Fig. 5. In this example, the approach can identify four latches ($P_R : \{L_1, L_2\}$, $R : \{L_9, L_{10}\}$) as the combinational dependent latches by selecting ($P_I : \{L_3\}$, $I : \{L_4, \ldots, L_8\}$) as the essential (substituted) latches. In addition, dependent latches' dependency functions can be explored with more accurate input support candidates as compared to the maximal input support candidates used in [15]. According to Theorem 4, the Boolean formulae of each dependency function can also be derived, as shown in the bottom of Fig. 5, by this heuristic.

Next, the physical meaning of the ordered destroyed cost is explained. It was observed that latches with similar behavior would have the same destroyed cost. Therefore, grouping the latches by the destroyed cost and consecutively moving L_i from the same group to P_I until this group becomes empty would speed up the identification process. Furthermore, dealing first with groups having a higher destroyed cost might identify more dependent latches. This is because latches in the higher cost group might have a higher probability of replacing more dependent latches. In this example, If L_3(dc_3=2) is moved to P_I, it can replace two latches, L_1 and L_2, with the independent latches $\{L_4, L_5\}$. However, if L_1(dc_1=1) is moved to P_I, only one latch, L_2 or L_3, could be replaced because the exact input support sets of L_2 and L_3 are $L_2 = \{L_3, L_4\}$ and $L_3 = \{L_1, L_2\}$, respectively. Therefore, the order in which latches are moved from P to P_I will influence the results obtained.

C. *Sequential functional dependency in the reachable state space*

To find more functional dependencies which exist in the reachable state space but not in the whole state space, an identification model with t timeframes is proposed, as shown in Fig. 6. To build this identification model, $(0, \ldots, t-1)$ circuit parts of the model in Fig. 1 are expanded by wiring internal PPOs and PPIs, and then connecting the same constraint part as that in Fig. 1 at the end of model. This model can detect the dependency for multiple timeframes. This functional dependency across multiple timeframes is called the *sequential functional dependency* in this paper.

To find the functional dependency at timeframe t, the identification model is first constructed as shown in Fig. 6. Then the same method as that mentioned in Section II is used to detect the dependencies in the constraint part of Fig. 6. In addition, given that a sequential system begins with an unrestricted initial state S_0, the reachable state space at each timeframe would be monotonically reduced and a fixed point of reachable state space would be reached. That is, $S_j \subseteq S_i$ if $j > i$ (S_t is the reachable state space at the t^{th} timeframe). Therefore, with the property of state space shrinking, if the functional dependency of L_i does not exist at timeframe t, it might exist at timeframe k ($k > t$). However, the functional dependency of L_i found at the timeframe t must still hold at the succeeding timeframe k ($k > t$).

Essential latch sets $P_I : \{L_3\}$, $I : \{L_4, L_5, L_6, L_7, L_8\}$
Dependent latch sets $P_R : \{L_1, L_2\}$, $R : \{L_9, L_{10}\}$

Corresponding input support candidates:
- L_1's input support candidates = $P_I \cup I$
- L_2's input support candidates = $P_I \cup I$
- L_9's input support candidates = I
- L_{10}'s input support candidates = I

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$ $\quad$ $L_9 = L_6 + L_7 + L_8$
$L_2 = L_3 \bullet \neg L_4$ $\quad$ $L_{10} = \neg(L_6 + L_7 + L_8)$

Fig. 5. The combinational dependent latches identified by this heuristic.

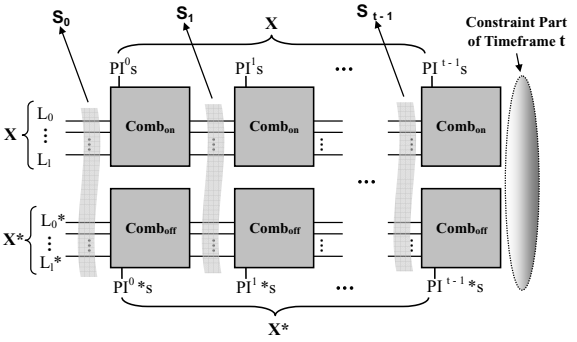

Fig. 6. The multi-timeframe model for sequential functional dependencies.

This approach will feed $P_I \cup I$, which is found at the current timeframe, to the next timeframe model to get more dependencies until run time limit is reached. If that is found, the previous dependency functions will be updated. Before feeding $P_I \cup I$ to the next timeframe model, a refined process will be conducted to examine whether latches in P_I are actually used by a dependent latch's dependency functions at the current timeframe. This is because that the heuristic might not obtain the optimal results every time, therefore, this process is used to refine the results. If some latches in P_I are not used, they will be removed from P_I to P and the operations mentioned above will be iterated. For the last example, $P_I \cup I = \{L_3\} \cup \{L_4, L_5, L_6, L_7, L_8\}$ is fed into the next timeframe model. Since L_3 in P_I is actually used by the derived dependency functions of L_1 and L_2, the refined process will be terminated. At the next timeframe, a new sequential dependency function ($L_6 = \neg L_7 \bullet L_8$) can be found. Hence, there is one more dependent latch L_6 after timeframe 2. The previous dependent latches' dependency functions which depend on L_6 are also updated. In this example, the dependency function of L_9 is updated from ($L_9 = L_6 + L_7 + L_8$) to ($L_9 = \neg L_7 \bullet L_8 + L_7 + L_8 = L_7 + L_8$) and the dependency function of L_{10} is updated from ($L_{10} = \neg(L_6 + L_7 + L_8)$) to ($L_{10} = \neg(\neg L_7 \bullet L_8 + L_7 + L_8) = \neg(L_7 + L_8)$) if L_6 is considered as a dependent latch after timeframe 2. The combinational and sequential dependent latches identified by this approach are summarized in Fig. 7.

Dependent latches: $\{L_1, L_2, L_9, L_{10}\}$ after timeframe 1

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$ $\quad$ $L_9 = L_6 + L_7 + L_8$
$L_2 = L_3 \bullet \neg L_4$ $\quad$ $L_{10} = \neg(L_6 + L_7 + L_8)$

Dependent latches: $\{L_1, L_2, L_6, L_9, L_{10}\}$ after timeframe 2

Corresponding dependency functions:

$L_1 = L_3 \bullet \neg L_5$ $\quad$ $L_9 = L_7 + L_8$
$L_2 = L_3 \bullet \neg L_4$ $\quad$ $L_{10} = \neg(L_7 + L_8)$
$L_6 = \neg L_7 \bullet L_8$

Fig. 7. The dependent latches identified at different timeframes by this approach.

The following paragraphs will explain an accelerated technique used for the sequential dependent latch identification and introduce the application of the found sequential functional dependencies.

*1) **Accelerated technique for the sequential dependent latch identification**:* The accelerated technique consists of two parts, the incremental technique [25] of SAT solvers and the early detection of dependent latches. To find the sequential functional dependency at timeframe t using SAT solvers, it needs to set the variables in the CNFs of circuit parts belonging to timeframe $(0, \ldots, t-1)$. Since these variables at timeframe $(0, \ldots, t-1)$ have been processed, much useful information is available to speed up the process at timeframe t. Thus, the heuristic uses the incremental technique of SAT solvers to reuse the learned clauses at timeframe $(0, \ldots, t-1)$. Furthermore, dependent latches existing at timeframe t' $(t' \geq 1)$ must also exist at timeframe t $(t > t')$, that is, sequential dependent latches existing at timeframe t $(t > 1)$ may have already existed at timeframe t' $(1 \leq t' \leq t-1)$. Therefore, these dependent latches detected earlier at timeframe t' $(1 \leq t' \leq t-1)$ are recorded and much run time could be saved in identifying all sequential dependent latches at timeframe t $(t > 1)$. In other words, to identify dependent latches at timeframe t, our approach would identify dependent latches from timeframe 1 to t, and only latches in $P_I \cup I$ at the current timeframe are needed for the next timeframe.

*2) **Application of sequential functional dependency**:* This heuristic can find the combinational functional dependency existing in the whole state space, as well as the sequential functional dependency existing in the reachable state space only. Therefore, an application of this work is to reduce the efforts of reachability analysis. With information about the sequential functional dependency at each timeframe, the reachability analysis engine which traverses the reachable state space of a sequential circuit can ignore some dependent latches after a certain timeframe. Hence, the searching space is significantly reduced. Furthermore, for a sequential circuit allowed to run t don't-care initializing cycles from unrestricted initial states before entering the normal operation states, the state space in timeframe $(0, \ldots, t-1)$ can be ignored (delay replaceability in [23]). Thus, the sequential functional dependencies found at timeframe t could be used to further optimize the design.

V. Experimental results

The proposed heuristic is implemented within SIS [22] environment and MiniSAT [8] is used as the SAT solver. The experiments are conducted on a Linux platform (CentOS 4.4) with a 2.194 GHz machine and 8GBytes memory. To show the scalability of the algorithm, a set of larger sequential circuits (more flip-flops (FFs)) are selected from ISCAS'89 and ITC'99 benchmarks.

The experimental results are shown in two subsections. Section V.A shows the capability of identifying dependent latches of our approach. Section V.B shows how dependent latch identification quantitatively benefits the reachability analysis. The reason that the proposed approach does not compare against [15] is that [15] reports the functional dependencies rather than the dependent latches. Therefore, only the results of the proposed approach are reported.

A. *Dependent latch identification by the proposed approach*

*1) **Combinational dependent latch identification**:* The experimental results of combinational dependent latches identified by this approach are shown in Table I. Columns 1 and 2 list the benchmarks and the number of FFs (also called latches). The results of this method are listed in Columns 3 to 6. |Dep. latch| represents the number of identified dependent latches. |Ess. latch| represents the number of essential latches which can be considered as substituted latches to replace the dependent latches. Column 5 shows the percentage of FFs which are identified as dependent latches (i.e., |Dep. latch| / |FF|). The final column is the CPU time of this approach measured in seconds.

For example, s13207 circuit has 669 FFs, in which this approach can identify 159 $(5 + 154)$ dependent latches and 510 $(453 + 57)$ essential latches. Therefore, in s13207 circuit, 23.77% of the latches in the circuit are the dependent latches. As demonstrated in Table I, this approach can recognize the dependent latches such that fewer latches have to be considered in the reachability analysis. In the experiment, we observed that the proposed approach identifies more dependent latches with the ISCAS benchmarks than with the ITC benchmarks. This indicates that fewer dependent latches are in the ITC benchmarks with unrestricted initial states.

TABLE I

Combinational dependent latch identification in **all state space**.

Circuit	|FF|	|Dep. latch| $R + P_R$	|Ess. latch| $I + P_I$	Dep. latch Ident. Ratio (%)	Time (s)
s5378	164	$1 + 7$	$135 + 21$	4.88	2.08
s9234	211	$1 + 19$	$165 + 26$	9.48	9.86
s15850	597	$8 + 14$	$568 + 7$	3.69	11.27
s13207	**669**	$\mathbf{5 + 154}$	$\mathbf{453 + 57}$	**23.77**	**283.18**
s38584	1452	$2 + 11$	$1428 + 11$	0.90	92.57
s38417	1636	$0 + 71$	$1541 + 24$	4.34	312.07
s35932	1728	$0 + 0$	$1728 + 0$	0	82.23
b12	121	$0 + 2$	$117 + 2$	1.65	0.11
b14	245	$2 + 0$	$243 + 0$	0.82	3.71
b15	449	$0 + 0$	$449 + 0$	0	5.79
b21	490	$4 + 0$	$486 + 0$	0.82	21.15
b22	735	$6 + 0$	$729 + 0$	0.82	67.74

TABLE II

Sequential dependent latch identification in **reachable state space** for $10,000$ seconds run time limit.

Circuit	|FF|	All State Space	Reachable State Space			Unfolded	All Dep.
		|Dep.|	|Dep.|	Timeframe	Ratio (%)	Timeframe	Ratio (%)
s5378	**164**	**8**	**53**	**14**	**27.44**	**106**	**32.32**
s9234	211	20	22	2	0.95	27	10.43
s15850	597	22	62	8	6.70	23	10.39
s13207	669	159	250	22	13.60	26	37.37
s38584	1452	13	32	4	1.31	7	2.20
s38417	1636	71	140	5	4.22	5	8.56
s35932	1728	0	0	0	0	9	0
b12	121	2	2	1	0	109	1.65
b14	245	2	2	1	0	27	0.82
b15	449	0	0	0	0	33	0
b21	490	4	4	1	0	13	0.82
b22	735	6	6	1	0	8	0.82

*2) **Sequential dependent latch identification**:* The experimental results on the sequential functional dependency in the reachable state space are shown in Table II. The CPU time limit is set to $10,000$ seconds. Columns 1 and 2 list the benchmarks and the number of FFs in it. Column 3 lists the number of identified dependent latches considered in all state space (i.e., at the timeframe 1). The results of sequential functional dependency are listed in Columns 4 through 6. Column 4 shows the number of dependent latches identified after the timeframe in Column 5. Column 6 shows the ratio of additionally identified sequential dependent latches as compared with total FFs in Column 2 (i.e., ((|Dep.| in Column 4) − (|Dep.| in Column 3)) / |FF|). Column 7 lists how many timeframes can be unfolded and examined within $10,000$ seconds run time limit. The ratio of the number of overall dependent latches identified after the timeframe in Column 5 is shown in the final column.

The experimental results show that other sequential functional dependencies may exist in the circuit, which can be used for reachability analysis and sequential circuit optimization. Take s5378 as an example: the state space could be reduced from 2^{164} to 2^{156} by ignoring the combinational dependent latches. Furthermore, the state space could be shrunk from 2^{156} to 2^{111} by ignoring other sequential dependent latches after the timeframe 14. However, after the timeframe 14, there are no additional sequential dependent latches can be identified. The reasons for it are the state space might has shrunk to a fixed point or some sequential dependent latches exist after the timeframe 107. But we cannot identify them with the

TABLE III

THE EXPERIMENTAL RESULTS OF REACHABILITY ANALYSIS WITHIN A RUN TIME LIMIT IN THE ORIGINAL CIRCUITS AND THE OPTIMIZED CIRCUITS.

Circuit	Time limit (s)	Ori.				Opt.				|RState| Improvement (Opt. / Ori.) (%)	BDD Size Reduction (%)
		|FF.|	Timeframe	|RState|	BDD size	|FF.|	Timeframe	|RState|	BDD size		
s5378	300,000	164	6	$6.77711e+16$	1,564,629	156	7	$1.63316e+17$	452,403	240.98	71.09
s9234	300,000	211	12	$4.64715e+17$	1,651,445	191	12	$4.64715e+17$	1,221,122	100	26.10
s15850	600,000	597	18	$7.21485e+07$	5,897,688	575	18	$7.21485e+07$	4,383,715	100	25.67
s13207	**600,000**	**669**	**43**	$\mathbf{4.82309e+15}$	**8,301,281**	**510**	**60**	$\mathbf{1.44247e+17}$	**10,762,172**	**2990.76**	**−29.64**
s38584	900,000	1,452	13	$1.61008e+10$	35,855,891	1,439	13	$1.61008e+10$	23,424,952	100	34.67
s38417	900,000	1,636	3	$3.45877e+18$	315,465	1,565	3	$3.45877e+18$	218,134	100	30.85

Note: The number of BDD size is reported from VIS.

time limit.

B. Reachability analysis enhancement with the dependent latch identification

This subsection provides the experimental results of the reachability analysis with and without using the results of the dependent latch identification within VIS [3] environment, the state-of-the-art academic formal verification tool. The experiments are conducted on a Linux platform (CentOS 4.4) with a 2.194 GHz machine and 8GBytes memory. The sequential circuits in ISCAS'89 which have dependent latches identified are selected as the benchmarks.

Although both the combinational and sequential dependent latches in a sequential circuit can be identified by our approach, the experiments only consider the combinational ones since VIS does not allow setting the sequential dependent latches during reachability analysis. The optimized version of one benchmark is derived by removing out the combinational dependent latches of the original circuit. The initial states of all latches in the optimized circuits are set to 0. But for the original circuits, only the essential latches are set to 0. The dependent latches in the original circuits are set to the values with respect to the dependency functions under the essential latches' setting. The dynamic variable ordering technique in BDD, sift algorithm, is used in the experiments (VIS command, "dynamic_var_ordering -e sift"). We conduct the experiments with a run time limit, and compare the results in term of the number of reached states and BDD size.

The experimental results of reachability analysis of the original and the optimized circuits with the same run time limit are shown in Table III. Columns 1 and 2 list the benchmarks and the corresponding run time limit. Longer run time limits are for more complicated circuits. The results of the original and the optimized benchmarks are listed in Columns 3 through 10. The timeframe column shows the timeframe reached within the run time limit. The number of reached states and the corresponding BDD size are listed in the next two columns. Column 11 lists the improvement ratio of the optimized circuit to the original one in term of the number of reached states. The final column lists the ratio of the BDD size reduction.

Take s13207 as an example, the original circuit has 669 FFs and the optimized circuit has 510 FFs by removing out 159 combinational dependent latches ($R + P_R = 5 + 154$). With 600,000 seconds run time limit, the original circuit reaches $4.82309e+15$ states in the 43^{th} timeframe while the optimized circuit reaches $1.44247e+17$ states in the 60^{th} timeframe. The fewer FFs in the optimized circuits made additional 17 timeframes evolved and 2890.76% (2990.76 − 100) more states reached. In addition, the optimized s5378 circuit also reached 140.98% (240.98 − 100) more states than the original one but had 71.09% BDD size reduction. For the circuit without state improvement, s9234, s15850, s38584 and s38417, the optimized versions still saved 26.10%, 25.67%, 34.67% and 30.85% BDD size, respectively. The BDD size savings make the reachability analysis proceed further possibly if the time limit is loosened.

VI. CONCLUSIONS

An algorithm has been proposed to efficiently identify combinational dependent latches and explore their dependency functions with the more accurate input support candidates. In addition, a multi-timeframe model was constructed to identify additional sequential dependent latches existing in the reachable state space. With the dependency functions identified, the complexity of performing reachability analysis and sequential depth exploration can be reduced by disregarding these dependent latches.

REFERENCES

[1] F. A. Aloul, I. L. Markov, et al, "Faster SAT and Smaller BDDs via Common Function Structure," Technical Report, University Michigan, 12, Dec. 2001.

[2] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. 35, pp. 677-691, August 1986.

[3] R. K. Brayton et al, "VIS: A System for Verification and Synthesis," in *Proc. Computer Aided Verification Conf.*, pp. 423-427, 1996.

[4] D. Brand, "Verification of Large Synthesized Designs," in *Proc. Int. Conf. Computer-Aided Design*, pp. 534-537, 1993.

[5] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," in *IEEE Trans. on Computer-Aided Design*, vol. 13, no. 4, pp. 401-424, April 1994.

[6] W. Craig, "Linear Reasoning: A New Form of the Herbrand-Gentzen Theorem," *J. Symbolic Logic*, 22(3):250-268, 1957.

[7] H. Cho et al, "Algorithms for Approximate FSM Traversal Based on State Space Decomposition," in *Proc. Design Automation Conf.*, pp. 25-30, 1993.

[8] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *Proc. Int. Conf. on Theory and Applications of Satisfiability Testing*, pp. 502-518, 2003.

[9] S. G. Govindaraju, D. L. Dill, A. Hu, and M. A. Horowitz, "Approximate Reachability Analysis with BDDs Using Overlapping Projections, in *Proc. Design Automation Conf.*, pp. 451-456 , 1998.

[10] A.-J. Hu and D. L. Dill, "Reducing BDD Size by Exploiting Functional Dependencies," in *Proc. Design Automation Conf.*, pp. 266-271, 1993.

[11] S.-Y. Huang, K.-T. Cheng, and K.-C. Chen, "AQUILA: An Equivalence Verifier for Large Sequential Circuits," in *Proc. Asian South Pacific Design Automation Conf.*, pp. 455-460, 1997.

[12] R. Hojati et al, "Heuristic Algorithms for Early Quantification and Partial Product Minimization," in Technical Report M94/11, UC Berkeley, 1994.

[13] J.-H. R. Jiang and R. K. Brayton, "Functional Dependency for Verification Reduction," in *Proc. Computer Aided Verification Conf.*, pp. 268-280, 2004.

[14] J. Krajicek, "Interpolation Theorems, Lower Bounds for Proof Systems, and Independence Results for Bounded Arithmetic," *J. Symbolic Logic*, 62(2):457-486, June 1997.

[15] C.-C. Lee, J.-H. R. Jiang, C.-Y. Huang, and A. Mishchenko, "Scalable Exploration of Functional Dependency by Interpolation and Incremental SAT Solving," in *Proc. Int. Conf. Computer-Aided Design*, pp. 227-233, 2007.

[16] B. Lin and A. R. Newton, "Exact Redundant State Registers Removal Based on Binary Decision Diagrams," in *Proc. Int. Conf. on Very Large Scale Integration*, pp. 277-286, 1991.

[17] M. Moskewicz, C. Madigan, L. Zhang, et al, "Chaff: Engineering an Efficient SAT Solver," in *Proc. Design Automation Conf.*, pp. 530-535, 2001.

[18] K. L. McMillan, "Interpolation and SAT-based Model Checking," in *Proc. Computer Aided Verification Conf.*, pp. 1-13, 2003.

[19] K. L. McMillan, "Symbolic Model Checking," Kluwer Academic Publishers, Norwell, MA, 1993.

[20] I.-H. Moon, et al, "Least Fixpoint Approximations for Reachability Analysis," in *Proc. Int. Conf. Computer-Aided Design*, pp. 41-49, 1999.

[21] P. Pudlak, "Lower Bounds for Resolution and Cutting Plane Proofs and Monotone Computations," *J. Symbolic Logic*, 62(3):981-998, Sep. 1997.

[22] E. M. Sentovich et al, "SIS: A System for Sequential Circuit Synthesis," Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[23] M. Syal and M. S. Hsiao, "VERISEC: Verifying Equivalence of Sequential Circuits Using SAT," in *Proc. High-Level Design Validation and Test Workshop.*, pp. 52-59, 2005.

[24] C. A. J. van Eijk et al, "Exploiting Functional Dependencies in Finite State Machine Verification," in *Proc. Euro. Design & Test Conf.*, pp. 9-14, 1996.

[25] J. Whittemore, J. Kim, and K. Sakallah, "SATIRE: A New Incremental Satisfiability Engine," in *Proc. Design Automation Conf.*, pp. 542-545, 2001.