A Halting Algorithm to Determine the Existence of Decoder

ShengYu Shen, Ying Qin, JianMin Zhang, and SiKun Li
School of Computer Science, National University of Defense Technology
410073,ChangSha,China
Email: {syshen,qy123,jmzhang,skli}@nudt.edu.cn

Abstract—Complementary synthesis automatically synthesizes the decoder circuit $E^{-1}$ of an encoder $E$. It determines the existence of $E^{-1}$ by checking the parameterized complementary condition (PC). However, this algorithm will not halt if $E^{-1}$ does not exist. To solve this problem, we propose a novel halting algorithm to check PC in two steps.

First, we over-approximate PC with the linear path unique condition (LP), and then falsify LP by searching for a loop-like path. If such a loop is found, then $E^{-1}$ does not exist; otherwise, LP can eventually be proved within $E$'s recurrence diameter.

Second, with LP proved above, we construct a list of approximations that forms an onion-ring between PC and LP. The existence of $E^{-1}$ can be proved by showing that $E$ belongs to all these rings.

To illustrate its usefulness, we have run our algorithm on several complex encoder circuits, including PCIE and 10G Ethernet. Experimental results show that our new algorithm always distinguishes correct Es from incorrect ones and halts properly.

Index Terms—Halting Algorithm, Complementary Synthesis

I. INTRODUCTION

Complementary synthesis has been proposed by us [1] to automatically synthesize an encoder circuit $E$'s decoder $E^{-1}$ in two steps. First, it determines the existence of $E^{-1}$ by checking the parameterized complementary condition (PC), i.e., whether $E$'s input can be uniquely determined by its output on a bounded unfolding of $E$’s transition function. Second, it builds $E^{-1}$ by characterizing its Boolean function with an all-solution SAT solver.

However, the bounded nature of the first step makes it an incomplete algorithm that will not halt if $E^{-1}$ does not exist.

To solve this problem, as shown in Figure 1, we propose a novel halting algorithm to check PC in two steps:

1) First, we over-approximate PC with the linear path unique condition (LP), i.e., every linear path of $E$ longer than a particular parameter $p$ always reaches the unique state set $S^U$, in which the input letter can be uniquely determined by the output letter, the current state and the next state. We then define the negative condition of LP, i.e., the loop-like non-unique condition (LL). We can falsify LP and prove LL by searching for a loop-like path that does not reach $S^U$ within $E$’s recurrence diameter $rd$. If we find such a loop-like path, then LL is proved and $E^{-1}$ does not exist; otherwise, a parameter $p$ can eventually be found to prove LP. In this case, we need the second step below to further check PC.

2) Second, with $p$ found in the first step that proves LP, we construct a list of approximations that forms an onion-ring between PC and its over-approximation LP. If $E$ is found in a certain ring but not in the next inner ring, then PC is falsified and $E^{-1}$ does not exist; otherwise, the existence of $E^{-1}$ is proved.

We have implemented our algorithm with the OCaml language, and solved the generated SAT instances with Zchaff SAT solver [2]. The benchmark set includes several complex encoders from industrial projects (e.g., PCIE and Ethernet), and their slightly modified variants without corresponding decoders. Experimental results show that our new algorithm always distinguishes correct encoders from their incorrect variants and halts properly. All experimental results and programs can be downloaded from http://www.ssypub.org.

This paper's contribution is: We propose the first halting algorithm to determines the existence of an encoder’s decoder.

The remainder of this paper is organized as follows. Section II presents background materials. Section IV introduces how to over-approximate PC with LP, and how to falsify LP by searching for loop-like paths. Section V discusses how to construct the onion-ring, and how to determine whether $E$ belongs to a certain ring. Section VI describes how to remove redundant output letters to minimize circuit area, while Section VII and VIII present experimental results and related works. Finally, Section IX concludes with a note on future work.

II. PRELIMINARIES

A. Basic Notation of Propositional Satisfiability Problem

For a Boolean formula $F$ over a variable set $V$, the Propositional Satisfiability Problem (abbreviated as SAT) is to find a satisfying assignment $A: V \rightarrow \{0,1\}$, so that $F$ can
be evaluated to 1. If such a satisfying assignment exists, then $F$ is **satisfiable**; otherwise, it is **unsatisfiable**.

A computer program that decides the existence of such a satisfying assignment is called a **SAT solver**, such as Zchaff [2], Grasp [3], Berkmin [4], and MiniSAT [5]. A formula to satisfying assignment is called a **SAT instance**.

**B. Recurrence Diameter**

A circuit can be modeled by **Kripke structure** $M = (S, I, T, A, L)$, with a finite state set $S$, the initial state set $I \subseteq S$, the transition relation $T \subseteq S \times S$, and the labeling of the states $L : S \rightarrow 2^A$ with atomic proposition set $A$.

Kroening et al. [6] defined the **state variables recurrence diameter** with respect to $M$, denoted by $\text{rrd}(M)$, as the longest loop-free path in $M$ starting from an initial state.

$$\text{rrd}(M) \overset{d}{=} \max \{ i | \exists s_0 \ldots s_i : I(s_0) \land \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \land \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \}$$

(1)

In this paper, we define a similar concept: the **uninitialized state variables recurrence diameter** with respect to $M$, denoted by $\text{uird}(M)$, is the longest loop-free path in $M$.

$$\text{uird}(M) \overset{d}{=} \max \{ i | \exists s_0 \ldots s_i : \bigwedge_{j=0}^{i-1} T(s_j, s_{j+1}) \land \bigwedge_{j=0}^{i-1} \bigwedge_{k=j+1}^{i} s_j \neq s_k \}$$

(2)

The only difference between these two definitions is that our $\text{uird}$ does not consider the initial state.

These definitions are only used in proving our theorems below. Our algorithm does not need to compute these diameters.

**C. The Original Algorithm to Determine the Existence of Decoder**

The complementary synthesis algorithm [1] includes two steps: determining the existence of decoder and characterizing its Boolean function. We will only introduce the first step here.

The encoder $E$ can be modeled by a Mealy finite state machine [7].

**Definition 1:** **Mealy finite state machine** is a 5-tuple $M = (S, s_0, I, O, T)$, consisting of a finite state set $S$, an initial state $s_0 \in S$, a finite set of input letters $I$, a finite set of output letters $O$, a transition function $T : S \times I \rightarrow S \times O$ that computes the next state and output letter from the current state and input letter.

As shown in Figure 2, as well as in the remainder of this paper, the state is represented as a gray round corner box, and the transition function $T$ is represented by a white rectangle.

We denote the state, input letter and output letter at the $n$-th cycle respectively as $s_n$, $i_n$ and $o_n$. We further denote the sequence of state, input letter and output letter from the $n$-th to the $m$-th cycle respectively as $s_n^m$, $i_n^m$ and $o_n^m$.

A sufficient condition for the existence of $E^{-1}$ is the **unique condition**, i.e., there exist two parameters $d$ and $l$, so that $i_n^m$ of $E$ can be uniquely determined by the output sequence $o_n^{m+d-l}$. As shown in Figure 3, $d$ is the relative delay between $o_n^{m+d-l}$ and the input letter $i_n$, while $l$ is the length of $o_n^{m+d-l}$.

However, the unique condition is unnecessarily restrictive, because it may not hold when $s_n$ is not reachable, even if $E$ is a correct encoder whose input can be uniquely determined by its output in its reachable state set. So we need to rule out unreachable states before checking the unique condition.

The continuous running character of communication circuits provides us an opportunity to rule out unreachable states easily without paying the expensive cost of computing the reachable state set. That is to say, we only need to check the unique condition on the state set $RS^\infty$ that can be reached infinitely often from $S$.

$$RS^q \overset{d}{=} \{ s_q | \bigwedge_{m=0}^{q-1} \{(s_{m+1}, o_m) \equiv T(s_m, i_m)\} \}$$

(3)

$$RS^{>p} \overset{d}{=} \bigcup_{q>p} RS^q$$

(4)

$$RS^\infty \overset{d}{=} \lim_{p \rightarrow \infty} RS^{>p}$$

(5)

Here, $RS^q$ is the set of states that can be reached from $S$ with exact $q$ steps.

According to Equation (5) and Figure 3, $RS^\infty$ can be easily over-approximated by prepending a state transition sequence of length $p$ to $s_n$, which forces $s_n$ to be in the state set $RS^{>p} = \bigcup_{q>p} RS^q$. Obviously, $RS^\infty$ and all $RS^{>p}$ form a total order shown below, which means a tighter over-approximation of $RS^\infty$ can be obtained by increasing the length $p$ of prepended state transition sequence.

---

Fig. 2. Mealy finite state machine

---

Fig. 3. The parameterized complementary condition
Thus, as shown in Figure 3, the parameterized complementary condition (PC) [1] can be defined as:

**Definition 2:** Parameterized Complementary Condition (PC) : For encoder $E$, $E \models PC(p, d, l)$ holds if $i_{n}$ can be uniquely determined by $o_{n+p-1}^{n+d-l}$ on $s_{n-p}^{n+d-1}$. This equals the unsatisfiability of $F_{PC}(p, d, l)$ in Equation (6). We further define $E \models PC$ as $\exists p, d, l : E \models PC(p, d, l)$.

$$
F_{PC}(p, d, l) \overset{def}{=} \begin{cases}
\bigwedge_{m=n-p}^{n+d-1} \{ (s_{m+1}, o_{m}) \equiv T(s_{m}, i_{m}) \} \\
\bigwedge_{m=n-p}^{n+d-1} \{ (s_{m+1}', o_{m}') \equiv T(s_{m}', i_{m}') \} \\
\bigwedge_{m=n-p}^{n+d-1} o_{m} \equiv o_{m}' \\
i_{n} \neq i_{a}'
\end{cases}
$$

(6)

The 2nd and 3rd lines of Equation (6) correspond respectively to two unfolded instances of $E$’s transition function. The only difference between them is that a prime is appended to every variable in the 3rd line. The 4th line forces the output sequences of these two unfolded instances to be the same, while the 5th line forces their input letters to be different.

III. Case Studies

To facilitate the understanding of our idea, we use some small examples shown in Figure 4, 5, and 6.

A. Case study 1

The circuit in Figure 4a) stores its input port $i_{n}$ in register $s_{n+1}$, and then outputs it to output port $o_{n+1}$. The unfolding of its transition function is shown in Figure 4b).

Obviously, $i_{n}$ is same as, and therefore can be uniquely determined by $s_{n+1}$. So $i_{n}$ can be uniquely determined by $s_{n}, o_{n}$, and $s_{n+1}$. So $LP$ is satisfied by this circuit. Here, the tuple $< s_{n}, o_{n}, s_{n+1} >$ can be seen as a ring that surrounds $i_{n}$.

Next, we expand the ring $< s_{n}, o_{n}, s_{n+1} >$ to another ring $< s_{n}, o_{n}, o_{n+1}, s_{n+2} >$, and perform the following 3 checks:

1) Whether $i_{n}$ can be uniquely determined by the ring $< s_{n}, o_{n}, o_{n+1}, s_{n+2} >$? Obviously the answer is yes.

2) Whether $i_{n}$ can be uniquely determined by $< o_{n}, o_{n+1}, s_{n+2} >$, the ring with $s_{n}$ removed? Obviously the answer is yes.

B. Case study 2

The circuit in Figure 5a) connects a constant 1, instead of input port $i$ to register $s$. So $i_{n}$ can never be determined by $s_{n}$, $o_{n}$, and $s_{n+1}$ in all states. Thus, a loop-like path with length 1 will reach such a state, which satisfies $LL$ and falsifies $LP$. So no decoder exists for this circuit.

C. Case study 3

For the circuit in Figure 6a), the unfolding of its transition function is shown in Figure 6b). It’s output is driven by constant 1, instead of register $s$. Obviously, this circuit can satisfy $LP$, which means $i_{n}$ can be uniquely determined by $s_{n}, o_{n}$, and $s_{n+1}$.

Next, we expand the ring $< s_{n}, o_{n}, s_{n+1} >$ to another ring $< s_{n}, o_{n}, o_{n+1}, s_{n+2} >$, and perform the following 3 checks:

1) Whether $i_{n}$ can be uniquely determined by the ring $< s_{n}, o_{n}, o_{n+1}, s_{n+2} >$? The answer is no, because $i_{n}$ never goto $o_{n}, o_{n+1}$ and $s_{n+2}$.

2) Whether $i_{n}$ can be uniquely determined by $< o_{n}, o_{n+1}, s_{n+2} >$, the ring with $s_{n}$ removed? The answer is still no with the same reason.

3) Whether $i_{n}$ can be uniquely determined by $< o_{n}, o_{n+1} >$, the ring with $s_{n}$ and $s_{n+2}$ both removed? The answer is still no with the same reason.

So in this case, no more expansion is needed, no decoder exists for this circuit.
and PC

Fig. 7. Two new uniqueness conditions

IV. OVER-APPROXIMATING PC WITH LP AND FALSIFYING LP BY SEARCHING FOR LOOP-LIKE PATH

A. Definition of Over-approximation

We first present some related definitions before defining the over-approximation of PC.

Definition 3: Unique State Set $SU$ and Non-unique State Set $SN$: For a circuit $E$, its unique state set $SU$ is the set of states $s_n$ that makes $i_n$ to be uniquely determined by $s_n, o_n$ and $s_{n+1}$, i.e., makes $F^U$ in Equation (7) unsatisfiable. The non-unique state set $SN$ is the complementary set of $SU$, i.e., $SN \overset{\Delta}{=} S - SU$.

\[
F^U \overset{\Delta}{=} \begin{cases} 
(s_{n+1}, o_n) \equiv T(s_n, i_n) \\
\land(s_{n+1}, o'_n) \equiv T(s'_n, i'_n) \\
\land o_n \equiv o'_n \land s_n \equiv s'_n \land s_{n+1} \equiv s'_{n+1} \\
\land i_n \neq i'_n
\end{cases}
\] (7)

To obtain a halting algorithm, we need to develop a negative condition for PC, which can recognize all those $E \models \neg PC$.

Unfortunately, it is very difficult, if not impossible, to develop such a condition. So we choose to first over-approximate PC with the linear path unique condition (LP), and then develop a negative condition for LP, i.e., the loop-like non-unique condition (LL). The definitions of LP and LL are given below, and presented intuitively in Figure 7a) and 7b).

Definition 4: Linear Path Unique Condition (LP): For encoder $E$, $E \models LP(p)$ holds if every linear path of length $p$ always reaches the unique state set $SU$. This equals the unsatisfiability of $F_{LP}(p)$ in Equation (8). We further define $E \models LP$ as $\exists p : E \models LP(p)$.

\[
F_{LP}(p) \overset{\Delta}{=} F^U \land \bigwedge_{m=n-p}^{n} \{(s_{m+1}, o_m) \equiv T(s_m, i_m)\}
\] (8)

Definition 5: Loop-like Non-unique Condition (LL): For encoder $E$, $E \models LL(p)$ holds if there exists a loop-like path of length $p$ that reaches the non-unique state set $SN$. This equals the satisfiability of $F_{LL}(p)$ in Equation (9). We further define $E \models LL$ as $\exists p : E \models LL(p)$.

\[
F_{LL}(p) \overset{\Delta}{=} F_{LP}(p) \land \bigvee_{m=n-p+1}^{n} \{ s_m \equiv s_{n-p} \}
\] (9)

Equation (9) is very similar to Equation (8), except that $\bigvee_{m=n-p+1}^{n} \{ s_m \equiv s_{n-p} \}$ is inserted to find a loop-like path.

The intuition behind LP and LL is to check whether $i_n$ can be uniquely determined by $s_n, s_{n+1}$ and $o_n$ with prepended $s_{n-p}$. Here, parameters $d$ and $l$ are removed, which makes it easier to find the value of $p$.

B. Relationships between PC, LP and LL

The relationships between PC, LP and LL are:

1. LP over-approximates PC, i.e., $E \models PC \Rightarrow E \models LP$.

2. Between LP and LL, there is always one and only one that holds, i.e., $E \models LP \Rightarrow E \models \neg LL$.

These relationships are presented intuitively in Figure 1, and their proofs are presented below. Those impatient readers can skip the remainder of this subsection.

Before proving these theorems, we need a lemma that defines a new formula for LP.

Lemma 1: With $F_{LP}(p, d, l)$ defined below:

\[
F_{LP}(p, d, l) \overset{\Delta}{=} \begin{cases} 
\bigwedge_{m=n-p}^{n+d-1} \{(s_{m+1}, o_m) \equiv T(s_m, i_m)\} \\
\land \bigwedge_{m=n+p}^{n+d-1} \{(s_{m+1}, o'_m) \equiv T(s'_m, i'_m)\} \\
\land o_n \equiv o'_n \land s_n \equiv s'_n \land s_{n+1} \equiv s'_{n+1} \\
\land i_n \neq i'_n
\end{cases}
\] (10)

we have: $F_{LP}(p, d, l) \iff F_{LP}(p)$

Proof: First, for the $\rightarrow$ direction. It is obvious that the clause set of $F_{LP}(p, d, l)$ is a superset of $F_{LP}(p)$, so the $\rightarrow$ direction is proved.

Second, to prove the $\leftarrow$ direction, we list below all additional sub-formulas that have been added into Equation (8) to obtain (10), and also our methods to satisfy them with a particular satisfying assignment $A$ of $F_{LP}(p)$.

1. $\bigwedge_{m=n-p}^{n} \{(s_{m+1}, o_m) \equiv T(s_m, i_m)\}$: This formula can be satisfied by assigning $A(s_m), A(i_m)$ and $A(o_m)$ to $s'_m, i'_m$ and $o'_m$ respectively.

2. $\bigwedge_{m=n+1}^{n+d-1} \{(s_{m+1}, o'_m) \equiv T(s'_m, i'_m)\}$: this formula represents a state transition sequence starting from $s_{n+1}$, which is satisfiable.

3. $\bigwedge_{m=n-p+1}^{n+d-1} \{(s_{m+1}, o'_m) \equiv T(s'_m, i'_m)\}$: this formula represents a state transition sequence starting from $s_{n-p+1}$, which is satisfiable with the same assignment defined in 2.

So, every satisfying assignment $A$ of $F_{LP}(p)$ can make $F_{LP}(p, d, l)$ satisfiable. So the $\leftarrow$ direction is proved. Thus, this theorem is proved.

In the remainder of this paper, we will use $F_{LP}(p)$ and $F_{LP}(p, d, l)$ interchangeably.

Theorem 1: $E \models PC(p, d, l) \rightarrow E \models LP(p)$

Proof: Let’s prove it by contradiction. Assume that $A : V \rightarrow \{0, 1\}$ is a satisfying assignment of $F_{LP}(p, d, l)$.

We define a new satisfying assignment $A'$ as:

\[
A'(v) \overset{\Delta}{=} \begin{cases} 
A(o_m) & v = o'_m, m \neq n \\
A(i_m) & v = i'_m, m \neq n \\
A(s_m) & v = s'_m, m \neq n \text{ and } m \neq n+1 \\
A(v) & \text{otherwise}
\end{cases}
\] (11)

Thus, $A'$ is also a satisfying assignment of $F_{LP}(p, d, l)$.

By comparing Equation (6) with (10), it is obvious that $A'$ is a satisfying assignment of the unsatisfiable formula $F_{PC}(p, d, l)$. 
This contradiction concludes the proof.

**Theorem 2**: \( E \equiv LP \iff E \equiv \neg LL \)

**Proof**: For the \( \rightarrow \) direction, let’s prove it by contradiction. Assume that \( E \equiv LL \). This means there exists a loop-like path that reaches state \( s_n \in S^N \).

Assume the length of this loop is \( q \), and the parameter of \( E \equiv LP \) is \( p \). Then we can unfold this loop \( [p/q] + 1 \) times, to get a path that is longer than \( p \) and reaches a state \( s_n \in S^N \). This will lead to \( E \equiv \neg LP(p) \).

This contradiction concludes the proof of the \( \rightarrow \) direction.

For the \( \leftarrow \) direction, assume that \( E \equiv \neg LP \) and \( E \equiv \neg LL \), then for all \( p \), \( F_{LP}(p) \) is satisfiable.

Assume the uninitialized state variables recurrence diameter of \( E \) is \( uirrd \), and let \( p = uirrd + 1 \). Then \( F_{LP}(p) \) is satisfiable, which means there is a path of length \( p \) that reaches a state \( s_n \in S^N \). Because \( p \) is larger than \( uirrd \), this path must contain a loop in it, which also makes \( F_{LL} \) satisfiable.

So \( E \equiv LL \) holds, which contradicts with \( E \equiv \neg LL \).

This contradiction concludes the proof of the \( \leftarrow \) direction.

C. Algorithm to Check \( E \equiv LP \) and \( E \equiv LL \)

Based on the relationships discussed in Subsection IV-B, we develop Algorithm 1 (as shown below) to check \( E \equiv LP \) and \( E \equiv LL \). This algorithm also discovers the value of parameter \( p \) if \( E \equiv LP \) holds.

```
Algorithm 1 checkLPLL
1: for \( p = 0 \to \infty \) do
2:     if \( F_{LP}(p) \) is unsatisfiable then
3:         print "\( E \equiv LP(p)\)"
4:         halt;
5:     else if \( F_{LL}(p) \) is satisfiable then
6:         print "\( E \equiv LL \) due to \( E \equiv LP(p)\)"
7:         halt;
8:     end if
9: end for
```

According to Theorem 2, Algorithm 1 will eventually halt at line 3 or 6 before \( p \) reaches \( E \)'s uninitialized state variables recurrence diameter \( uirrd \). Thus, we have the following theorem.

**Theorem 3**: Algorithm 1 is a halting algorithm.

With Algorithm 1, we can determine whether \( E \) is an improperly designed encoder that leads to \( E \equiv LL \). But if \( E \equiv LP \), how to determine whether \( E \) is a correct encoder that leads to \( E \equiv PC \)? We will discuss this problem in the next section.

V. CHECKING \( E \equiv PC \) BY CONSTRUCTING ONION-RING

A. Intuitive Description

To make it easier to follow our presentation, we present our idea intuitively here with an example.

As shown in Figure 8, we add two new parameters \( b \) and \( f \) to replace \( d \) and \( l \). The backward parameter \( b \) refers to the distance between state \( s_{n-b} \) and \( s_n \). The forward parameter \( f \) refers to the distance between state \( s_{n+1} \) and \( s_{n+1+f} \). The relations between \( b, f \) and \( d, l \) are:

\[
\begin{align*}
    d &= f \\
    l &= b + f + 1
\end{align*}
\]

Because Algorithm 1 already recognizes all \( E \)'s that lead to \( E \equiv LL \), we only need to deal with those \( E \)'s that lead to \( E \equiv LP(p) \) here. This will result in the following proposition:

**Proposition 1**: \( i_n \) is uniquely determined by \( s_n \), \( o_n \) and \( s_{n+1} \).

As shown in Figure 8, we can further generalize Proposition 1 by:

1) Replacing \( o_n \) with \( o_{n-f} \).
2) Replacing \( s_n \) with \( s_{n-b} \).
3) Replacing \( s_{n+1} \) with \( s_{n+f+1} \).

and thus obtain:

**Proposition 2**: \( i_n \) is uniquely determined by \( s_{n-b} \), \( o_{n-b} \) and \( s_{n+f+1} \).

It is obvious that Proposition 1 is a special case of Proposition 2, with \( b = 0 \) and \( f = 0 \).

With this generalization, our algorithm will be intuitively described in the following five steps:

1) First, we ignore both \( s_{n-b} \) and \( s_{n+f+1} \), and test whether \( i_n \) can be uniquely determined by \( o_{n-b} \). If yes, our algorithm halts with \( E \equiv PC \).
2) Otherwise, we ignore \( s_{n-b} \), and test whether \( i_n \) can be uniquely determined by \( o_{n-b} \) and \( s_{n+1} \). If yes, then \( i_n \) definitely does NOT depend on any \( o_k \) with \( k < n-b \), but it may still depend on some \( o_k \) with \( k > n + f \). So we need to increase \( f \) by 1 and goto step 1.
3) Otherwise, we ignore \( s_{n+f+1} \), and test whether \( i_n \) can be uniquely determined by \( s_{n-b} \) and \( o_{n-b} \). If yes, then \( i_n \) definitely does NOT depend on any \( o_k \) with \( k > n + f \), but it may still depend on some \( o_k \) with \( k < n-b \). So we need to increase \( b \) by 1 and goto step 1.
4) Otherwise, we test whether \( i_n \) can be uniquely determined by \( s_{n-b} \), \( o_{n-b} \) and \( s_{n+1} \). If yes, then \( i_n \) may depend on some \( o_k \) with both \( k < n-b \) and \( k > n + f \), so we need to increase \( b \) and \( f \) by 1, and goto step 1.
5) If the algorithm reaches here, then \( i_n \) had been uniquely determined by \( s_{n-b} \), \( o_{n-b} \) and \( s_{n+f+1} \) previously, but NOT by \( s_{n-b} \), \( o_{n-b} \) and \( s_{n+1} \) now, where \( b' \leq b \) and \( f' \leq f \). This means that adding more \( o_k \) into \( o_{n-b} \) by

![Fig. 8. Forward and backward constraints](image-url)
increasing \( b \) and \( f \), will never make \( PC \) holds. Our algorithm halts with \( E \models \neg PC \).

In this algorithm, every step with a pair of \( b \) and \( f \) corresponds to an onion-ring defined in subsection V-B. If it halts at step 5, then \( E \) belongs to the ring corresponding to \( b' \) and \( f' \), but does not belong to the next inner ring corresponding to \( b \) and \( f \), which means \( E^{-1} \) does not exist.

Formal presentation and proof will be given in the next two subsections.

B. Constructing Onion-Ring between PC and LP

According to Figure 8, we define the following formulas: \( F_{\text{unfold}} \) defines two unfolded instances of transition function, and constrains that their output sequence are equivalent, whereas their input letters are inequivalent:

\[
F_{\text{unfold}}(p, b, f) \overset{\text{def}}{=} \begin{cases} 
\wedge_{m=n-p}^{n+f} \{ (s_{m+1}, o_m) \models T(s_m, i_m) \} \\
\wedge_{m=n-p}^{n+f} \{ (s'_{m+1}, o'_m) \models T(s'_m, i'_m) \} \\
\wedge_{m=n-b}^{n+f} \{ o_m = o'_m \} 
\end{cases}
\] (13)

\( F_{\text{backward}} \) constrains that \( s_{n-b} \) equals \( s'_{n-b} \):

\[
F_{\text{backward}}(p, b, f) \overset{\text{def}}{=} \{ s_{n-b} = s'_{n-b} \}
\] (14)

\( F_{\text{forward}} \) constrains that \( s_{n+1+f} \) equals \( s'_{n+1+f} \):

\[
F_{\text{forward}}(p, b, f) \overset{\text{def}}{=} \{ s_{n+1+f} = s'_{n+1+f} \}
\] (15)

With these formulas, we define 4 new unique conditions between \( PC \) and \( LP \).

\( LP_{\text{nobf}}(p, b, f) \): \( i_n \) can be uniquely determined by \( o_{n-b}^{+} \). This equals the unsatisfiability of \( F_{LP_{\text{nobf}}} \) in Equation (16).

\[
F_{LP_{\text{nobf}}}(p, b, f) \overset{\text{def}}{=} F_{\text{unfold}}
\] (16)

\( LP_{f}(p, b, f) \): \( i_n \) can be uniquely determined by \( o_{n-b}^{+} \) and \( s_{n+1+f} \). This equals the unsatisfiability \( F_{LP_{f}} \) in Equation (17).

\[
F_{LP_{f}}(p, b, f) \overset{\text{def}}{=} F_{\text{unfold}} \land F_{\text{forward}}
\] (17)

\( LP_{b}(p, b, f) \): \( i_n \) can be uniquely determined by \( s_{n-b} \) and \( o_{n-b}^{+} \). This equals the unsatisfiability \( F_{LP_{b}} \) in Equation (18).

\[
F_{LP_{b}}(p, b, f) \overset{\text{def}}{=} F_{\text{unfold}} \land F_{\text{backward}}
\] (18)

\( LP_{bf}(p, b, f) \): \( i_n \) can be uniquely determined by \( s_{n-b} \), \( o_{n-b}^{+} \) and \( s_{n+1+f} \). This equals the unsatisfiability \( F_{LP_{bf}} \) in Equation (19).

\[
F_{LP_{bf}}(p, b, f) \overset{\text{def}}{=} F_{\text{unfold}} \land F_{\text{backward}} \land F_{\text{forward}}
\] (19)

These new unique conditions, when coupled with parameters \( b \) and \( f \), will act as onion-rings between \( PC \) and its over-approximation \( LP \).

It is obvious that, \( LP_{bf} \) is very similar to \( LP \), while \( LP_{\text{nobf}} \) is very similar to \( PC \). Such similarities will be employed to prove the correctness of our approach.

Some lemmas that will be used to prove the correctness of the onion-ring approach are given below.

**Lemma 2**: \( E \models LP_{bf}(p, b, f) \iff E \models LP_{bf}(p, b, f+1) \)

**Proof**: Let’s prove it by contradiction. Assume that \( E \models \neg LP_{bf}(p, b, f) \) and \( E \models LP_{bf}(p, b, f+1) \), which means that \( F_{LP_{bf}}(p, b, f) \) is satisfiable while \( F_{LP_{bf}}(p, b, f+1) \) is unsatisfiable.

We can append a state transition to \( F_{LP_{bf}}(p, b, f) \) after \( s_{n+f+1} \), and get a new formula \( F'_{LP_{bf}}(p, b, f) \).

\[
F'_{LP_{bf}}(p, b, f) \overset{\text{def}}{=} F_{LP_{bf}}(p, b, f) \\
\land (s_{n+f+2}, o_{n+f+1}) = T(s_{n+f+1}, i_{n+f+1})
\] (20)

This newly appended state transition is satisfiable, which makes \( F'_{LP_{bf}} \) a satisfiable formula.

Assume \( A \) is a satisfying assignment of \( F'_{LP_{bf}}(p, b, f) \). We define another satisfying assignment \( A' \) as:

\[
A'(v) \overset{\text{def}}{=} \begin{cases} 
A(o_{n+f+1}) & v = o_{n+f+1} \\
A(i_{n+f+1}) & v = i_{n+f+1} \\
A(s_{n+f+2}) & v = s_{n+f+2} \\
A(v) & \text{otherwise}
\end{cases}
\] (21)

Obviously, \( A' \) is a satisfying assignment of unsatisfiable formula \( F_{LP_{bf}}(p, b, f+1) \).

This contradiction concludes the proof.

**Lemma 3**: \( E \models LP_{b}(p, b, f) \iff E \models LP_{b}(p, b+1, f) \)

Its proof is similar to that of Lemma 2.

**Lemma 4**: If \( E \models PC(p, d, l) \), then the following Equation holds:

\[
E \models LP_{bf} \quad (p + d - l + 1, 0, d)
\] (22)

**Proof**: For the \( \rightarrow \) direction, according to Equation (18) and (19), it is obvious that the clause set of \( F_{LP_{bf}} \) is a superset of \( F_{LP_{b}} \), which means the unsatisfiability of the latter one implies the unsatisfiability of the former one. So the \( \rightarrow \) direction is proved.

For the \( \leftarrow \) direction, let’s prove it by contradiction. Assume that \( F_{LP_{bf}} \) is unsatisfiable, and \( A \) is a satisfying assignment of \( F_{LP_{bf}} \). We define another assignment \( A' \) as:

\[
A'(v) \overset{\text{def}}{=} \begin{cases} 
A(o_k) & v = o_k \text{ where } k < n \\
A(i_k) & v = i_k \text{ where } k < n \\
A(s_k) & v = s_k \text{ where } k < n \\
A(v) & \text{otherwise}
\end{cases}
\] (23)

Obviously, \( A' \) is a satisfying assignment of Equation (6), which means \( E \not\models PC(p, d, l) \). This contradiction concludes the proof of the \( \leftarrow \) direction.

**Lemma 5**: If \( E \models PC(p, d, l) \), then the following Equation holds:

\[
E \models LP_{\text{bf}} \quad (p, l - d - 1, d)
\] (24)

Its proof is similar to that of Lemma 4.
An important theorem that defines the onion-ring is presented and proved below:

**Theorem 4:** If $E \models PC(p, d, l)$, then there exists a list of unique conditions with their relationship shown below:

\[ E \models LP \quad (p + d - l + 1) \]
\[ \vdash E \models LP_{bf} \quad (p + d - l + 1, 0, 0) \]
\[ \vdash E \models LP_{bf} \quad (p + d - l + 1, 0, 1) \]
\[ \vdots \]
\[ \vdash E \models LP_{bf} \quad (p + d - l + 1, 0, d - 1) \]
\[ \vdash E \models LP_b \quad (p + d - l + 1, d) \]
\[ \vdash E \models LP_b \quad (p + d - l + 2, 1, d) \]
\[ \vdots \]
\[ \vdash E \models LP_b \quad (p - l - d - 1, d) \]
\[ \vdash E \models LP_{nobl} \quad (p - l - d - 1, d) \]
\[ \vdash E \models PC \quad (p, d, l) \]

**Proof:** According to Equation (10) and (19), the relation between the 1st and 2nd line of Equation (25) holds.
According to Lemma 2, the relations between the 2nd and 6th line of Equation (25) holds.
According to Lemma 4, the relation between the 6th and 7th line of Equation (25) holds.
According to Lemma 3, the relation between the 7th and 10th line of Equation (25) holds.
According to Lemma 5, the relation between the 10th and 11th line of Equation (25) holds.
According to Equation (6) and (16), the relations between the last two lines of Equation (25) holds.

In Equation (25), all $\vdash$ symbols form a total order, which makes all unique conditions on the right-hand side of $\models$ to form an onion-ring (as shown in Figure 1).

**C. Algorithm Implementation**

With those theorems presented in Subsection V-B, we use the following Algorithm 2 to check $E \models PC$.

**Algorithm 2** checkPCLP($p, b, f$)

1: if $F_{LP_{nobl}}(p, b, f)$ is unsatisfiable then
2: \hspace{1cm} print "$E \models PC(p, f, b + f + 1)$"
3: \hspace{1cm} halt;
4: else if $F_{LP_{bf}}(p, b, f)$ is unsatisfiable then
5: \hspace{1cm} checkPCLP($p, b, f + 1$)
6: else if $F_{LP_b}(p, b, f)$ is unsatisfiable then
7: \hspace{1cm} checkPCLP($p + 1, b + 1, f$)
8: else if $F_{LP_{2f}}(p, b, f)$ is unsatisfiable then
9: \hspace{1cm} checkPCLP($p + 1, b + 1, f + 1$)
10: else
11: \hspace{1cm} print "no $E^{-1}$ due to $E \models \neg PC$"
12: \hspace{1cm} halt;
13: end if

Algorithm 2 is invoked with the form checkPCLP($p, 0, 0$), with $p$ computed by Algorithm 1.

Algorithm 2 just follows the onion-ring defined by Equation (25), from the first line to the last line. If $E \models PC$ holds, it will eventually reach line 2, and the existence of $E^{-1}$ is proved; otherwise, it will eventually reach line 11, which means that $E$ does not belong to the current ring, and $PC$ is falsified. So $E^{-1}$ does not exist.

Thus, with Theorem 4, we have the following theorem.

**Theorem 5:** Algorithm 2 is a halting algorithm.

**VI. REMOVING REDUNDANT OUTPUT LETTERS**

Although Algorithm 1 and 2 together are sufficient to determine the existence of $E^{-1}$, the parameters found by line 2 of Algorithm 2 contain some redundancy, which will cause unnecessary large overhead of circuit area.

For example, as shown in Figure 9, assume that $l$ is the smallest parameter value that leads to $E \models PC(p, d, l)$, and $l < d$, which means that $i_n$ is uniquely determined by some output letters $o_k$ with $k > n$.

We further assume that line 2 of Algorithm 2 find out $E \models PC(p, d, l')$. It is obvious that $l' > d$, which make $i_n$ to depend on some redundant $o_k$ with $k \leq n$.

So $o_{n+d-l'-1}$ is the sequence of redundant output letters, which should be removed to prevent them from being instantiated as latches in circuit $E^{-1}$.

Algorithm 3 that removes these redundant output letters is presented below:

**Algorithm 3** RemoveRedundancy($p, d, l'$)

1: $l = 0 \rightarrow l'$
2: if $F_{PC(p, d, l)}$ is unsatisfiable then
3: \hspace{1cm} print "$E \models PC(p, d, l)"
4: \hspace{1cm} halt;
5: end if
6: end for

**VII. EXPERIMENTAL RESULTS**

We have implemented our algorithm in Zchaff [2], and run it on a PC with a 2.4GHz Intel Core 2 Q6600 processor, 8GB memory and CentOS 5.2 linux. All experimental results and programs can be downloaded from http://www.ssypub.org.

**A. Benchmarks**

Table I shows information of the following benchmarks.
1) A XGXS encoder compliant to clause 48 of IEEE-802.3ae 2002 standard [8].

![Fig. 9. Redundant Output Letters](image-url)
2) A XFI encoder compliant to clause 49 of the same IEEE standard.

3) A 66-bit scrambler used to ensure that a data sequence has sufficiently many 0-1 transitions, so that it can run through high-speed noisy serial transmission channel.

4) A PCIE physical coding module.

5) The Ethernet module of Sun’s OpenSparc T2 processor.

B. Experimental Results on Properly Designed Encoders

The 2nd and 6th rows of Table II compares the run time of checking $E \models PC$ between [1] and our approach. The run time of our approach are much larger than [1]. This is caused by checking the unique and non-unique conditions defined in Section IV and V.

The 3rd and 7th rows compare the discovered parameter values, and some minor differences are found on parameter $p$. This is caused by the different orders in checking various parameter combinations.

According to [9], $p$ is used to constrain the reachable states, while $d$ and $l$ will affect the run time of building $E^{-1}$ and its circuit area. To prove this, we compared the run time of building $E^{-1}$ with all-solution SAT solver in the 4th and 8th rows of Table II, and also compared the area of $E^{-1}$ in the 5th and 9th rows. These $E^{-1}$s were synthesized with DesignCompiler and LSI10 target library.

The results indicate that the differences in parameter $p$ do not cause significant overhead in the run time of all-solution SAT solver and circuit area.

C. Experimental Results on Improperly Designed Encoders

To further show the usefulness of our algorithm, we need some improperly designed encoders without corresponding decoders.

We obtained these improperly designed encoders by modifying each benchmark’s output statements, such that they can explicitly output the same letter for two different input letters. In this way, input letter $i_n$ can never be uniquely determined by $E$’s output sequence.

The 2nd row of Table III shows the result of Algorithm 1, while the 3rd row shows the result of Algorithm 2. The total run time is shown in the 4th row.

For XFI and scrambler, the result of Algorithm 1 is $LL$, which falsifies $PC$ directly. So the result of Algorithm 2 is NA.

The results indicate that our algorithm always terminated, and recognized these modified incorrect encoders.

VIII. Related Works

A. Complementary Synthesis

The concept of complementary synthesis was first proposed by us [1] in ICCAD 2009. Its major shortcomings are that it is incomplete, and its run-time overhead of building complementary circuit is too large.

The incomplete problem has been addressed by this paper, while we [9] addresses the second shortcoming by simplifying the SAT instance with unsatisfiable core extraction before building complementary circuits.

B. The Completeness of Bounded Model Checking

Bounded model checking (BMC) is a model checking technology that considers only those paths of limited length. Many researchers try to find out complete approaches for BMC.

One line of research [6], [10] tries to find out a bound $b$, which can guarantee the correctness of a specification on all paths, if the specification is correct on all paths shorter than $b$.

The other line of research [11] tries to find out a pattern for induction, such that the correctness of a specification within any bound $b$ implies the correctness on bound $b + 1$.

Our approach achieves completeness without following these two approaches. Instead, we define two complement uniqueness conditions, $LP$ and $LL$, and find out proper algorithms to check them.

C. Temporal Logic Synthesis

The temporal logic synthesis was first addressed by Clarke et.al [12] and Manna et.al [13]. But Pnueli et.al [14] pointed out that the complexity of LTL synthesis is double exponential.

One line of research [15]–[17] focuses on the so-called generalized reactive formulas of the form: $(\square q_1 \land \cdots \land \square q_m) \rightarrow (\square q_1 \land \cdots \square q_n)$. Complexity of solving synthesis problem for such formula is $O(N^2)$. 

<table>
<thead>
<tr>
<th>Line number of verilog source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGXS</td>
</tr>
<tr>
<td>214</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data path width</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGXS</td>
</tr>
<tr>
<td>13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg 1 result</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGXS</td>
</tr>
<tr>
<td>LP(1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alg 2 result</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGXS</td>
</tr>
<tr>
<td>~PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time(sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XGXS</td>
</tr>
<tr>
<td>1.23</td>
</tr>
</tbody>
</table>

We obtained these improperly designed encoders by modifying each benchmark’s output statements, such that they can explicitly output the same letter for two different input letters. In this way, input letter $i_n$ can never be uniquely determined by $E$’s output sequence.

The 2nd row of Table III shows the result of Algorithm 1, while the 3rd row shows the result of Algorithm 2. The total run time is shown in the 4th row.

For XFI and scrambler, the result of Algorithm 1 is $LL$, which falsifies $PC$ directly. So the result of Algorithm 2 is NA.

The results indicate that our algorithm always terminated, and recognized these modified incorrect encoders.
The other line of research focuses on finding efficient algorithm [18] for expensive safra determination algorithm [19] on an useful formula subset, or just avoiding it [20].

Based on these research works, some tools [21] that can handle small temporal formulas have been developed. All these works assume a hostile environment, which seems too restrictive for many applications. So Fisman et.al [22], Chatterjee et.al [23] and Ummels et.al [24] proposed rational synthesis algorithm, which assumes that each agents act to achieve their own goals instead of failing each other.

D. Protocol Converter Synthesis

The protocol converter synthesis was first proposed by Avnit et.al [25] to automatically generate a translator between two different communication protocols. Avnit et.al [26] improved it with a more efficient design space exploration algorithm. The implementation of this tool is introduced in [27].

IX. CONCLUSIONS AND FUTURE WORKS

This paper proposes the first halting algorithm that checks whether a particular encoder $E$ has corresponding decoder. Theoretical analysis and experimental results show that our approach always distinguishes correct encoders from their incorrect variants and halts properly.

One future work is to develop a debugging method to find out why $E^{-1}$ does not exist. For the failure caused by loop-like path, we plan to develop a debugging mechanism based on our previous work on loop-like counterexample minimization [28].

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for their hard work.

This work was fund by project 60603088 supported by National Natural Science Foundation of China.

This work was also supported by the Program for Changjiang Scholars and Innovative Research Team in University No IRT0614.

REFERENCES