Transactions on Design
Automation of Electronic Systems

# Complementary Synthesis for Encoder with Flow Control Mechanism

| | |
|---|---|
| Journal: | *Transactions on Design Automation of Electronic Systems* |
| Manuscript ID: | TODAES-2014-P-793.R2 |
| Manuscript Type: | Paper |
| Date Submitted by the Author: | n/a |
| Complete List of Authors: | Qin, Ying; National University of Defense Technology, School of Computer<br>Shen, ShengYu; National University of Defense Technology, School of Computer<br>Wu, QingBo; National University of Defense Technology, School of Computer<br>Dai, HuaDong; National University of Defense Technology, School of Computer<br>Yan, Jia; National University of Defense Technology, School of Computer |
| Computing Classification Systems : | B.5.2 [Design Aids]: Automatic synthesis, B.6.3 [Design Aids]: Automatic synthesis |
| | |

# Complementary Synthesis for Encoder with Flow Control Mechanism

YING QIN , SHENGYU SHEN , QINGBO WU , HUADONG DAI and YAN JIA, School of Computer, National University of Defense Technology

Complementary synthesis automatically generates an encoder's decoder with the assumption that the encoder's all input variables can always be uniquely determined by its output symbol sequence. However, to prevent the faster encoder from overwhelming the slower decoder, many encoders employ flow control mechanism that fails this assumption. Such encoders, when their output symbol sequences are too fast to be processed by the decoders, will stop transmitting data symbols, but instead transmitting idle symbols that can only uniquely determine a subset of the encoder's input variables. And the decoder should recognize and discard these idle symbols. This mechanism fails the assumption of all complementary synthesis algorithms, because some input variables can't be uniquely determined by the idle symbol.

A novel algorithm is proposed to handle such encoders. First, it identifies all input variables that can be uniquely determined, and takes them as flow control variables. Second, it infers a predicate over these flow control variables that enables all other input variables to be uniquely determined. Third, it characterizes the decoder's Boolean function with Craig interpolant.

Experimental results on several complex encoders indicate that this algorithm can always correctly identify the flow control variables, infer the predicates and generate the decoder's Boolean functions.

Categories and Subject Descriptors: B.5.2 [**Design Aids**]: Automatic synthesis; B.6.3 [**Design Aids**]: Automatic synthesis

General Terms: Algorithms, Verification

Additional Key Words and Phrases: Complementary synthesis, Craig interpolant, decoder, encoder, finite-state transition system, satisfiability solving

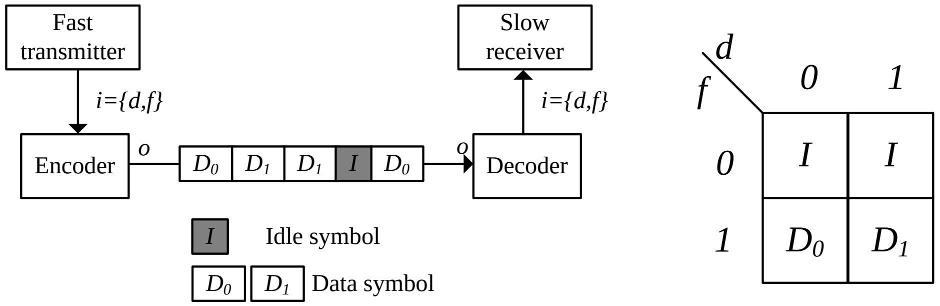
**ACM Reference Format:**
Ying Qin , ShengYu Shen , QingBo Wu , HuaDong Dai and Yan Jia, 2014. Complementary Synthesis for Encoder with Flow Control Mechanism. *ACM Trans. Des. Autom. Electron. Syst.* 9, 4, Article 39 (March 2010), 25 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

## 1. INTRODUCTION

One of the most difficult jobs in designing communication and multimedia chips is to design and verify complex encoder and decoder pairs. The encoder maps its input variables $\vec{i}$ to its output variables $\vec{o}$, while the decoder recovers $\vec{i}$ from $\vec{o}$. Complementary synthesis [Shen et al. 2009;2010; 2011; 2012;Liu et al. 2011;2012;Tu and Jiang 2013] eases this job by automatically generating a decoder from an encoder, with the assumption that $\vec{i}$ can always be uniquely determined by a bounded sequence of $\vec{o}$.

This work was funded by projects 61070132 and 61133007 supported by National Natural Science Foundation of China, the 863 Project of China under contract 2013AA014301, project 4345135127 supported by Program for New Century Excellent Talents in University.
Author's addresses: Ying Qin, ShengYu Shen, QingBo Wu, HuaDong Dai and Yan Jia , School of Computer, National University of Defense Technology.

DOI:http://dx.doi.org/10.1145/0000000.0000000



| f \ d | 0 | 1 |
|---|---|---|
| 0 | $I$ | $I$ |
| 1 | $D_0$ | $D_1$ |

a) The faster transmitter and slower receiver connected by the encoder and decoder

b) The mapping table of the encoder with flow control mechanism that maps *d* and *f* to *o*

Fig. 1. An encoder with flow control mechanism

However, the encoders of many high speed communication systems employ flow control mechanism [Abts and Kim 2011] that fails this assumption. Figure 1a) shows such a communication system with flow control mechanism, which includes a faster transmitter and a slower receiver connected by a pair of encoder and decoder. There are two input variables from the transmitter to the encoder: the data bit $d$ to be encoded, and the flow control bit $f$ indicating the validness of $d$. Figure 1b) shows the mapping table of the encoder that maps $f$ and $d$ to the output symbol $\vec{o}$.

The flow control mechanism prevents the faster transmitter from overwhelming the slower receiver in the following way:

(1) When the receiver can keep up with the transmitter, the transmitter will rise $f$ to 1, which makes the encoder to transmit $D_d$ according to the value of $d$. And the decoder can always recover both $f$ and $d$ from $D_d$ according to Figure 1b).
(2) But when the receiver can't keep up with the transmitter, the transmitter will drop $f$ to 0 to stop the encoder transmitting new $D_d$, but instead transmitting the idle symbol $I$ without considering the value of $d$. And the decoder should discard this idle symbol $I$, and send $f \equiv 0$ to the receiver with whatever value on $d$.

However, this mechanism fails the assumption of all complementary synthesis algorithms, because $d$ can't be uniquely determined by the idle symbol $I$. So to resolve this issue, we only need to consider the case $f \equiv 1$, in which $d$ can be uniquely determined. For other cases in which $f \equiv 0$, $d$ isn't needed by the receiver. So, we propose a novel complementary synthesis algorithm to handle flow control mechanism :

(1) First, it applies the classical halting complementary synthesis algorithm [Shen et al. 2011] to identify all the input variables of the encoder that can be uniquely determined, and call them the flow control variables $\vec{f}$. Other input variables that can't be uniquely determined are called the data variables $\vec{d}$.
(2) Second, it infers a sufficient and necessary predicate $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined by a bounded sequence of the encoder's output variables $\vec{o}$.
(3) Finally, it characterizes the decoder's Boolean function that computes each flow control variable $f \in \vec{f}$ by building a Craig interpolant [McMillan 2003]. On the other hand, for other data variables $\vec{d}$, their values are meaningful only when $valid(\vec{f}) \equiv 1$. Thus, the decoder's Boolean function of each $d \in \vec{d}$ can be built similarly with Craig interpolant, but only after enforcing $valid(\vec{f}) \equiv 1$.

The second step of this algorithm seems somewhat similar to that of [Shen et al. 2012] in the sense that both algorithms infer predicates that enable $\vec{d}$ or $\vec{i}$ to be uniquely determined. But the essential difference between them is that the algorithm of [Shen et al. 2012] infers a global assertion that must be enforced on all steps along the unrolled state transition sequence, while our algorithm infers a local predicate that is only enforced at the current step when we need to recover the value of $\vec{d}$. Thus, our algorithm can be seen as a generalization of [Shen et al. 2012].

Experimental results indicate that, for several complex encoders (e.g., Ethernet [IEEE 2012] and PCI Express [PCI-SIG 2009]), our algorithms can always correctly identify the flow control variables, infer the predicates and generate the decoders. In addition, we also conduct other experiments to compare our algorithms with the state-of-the-art complementary synthesis algorithms. All these experimental results and programs can be downloaded from https://github.com/shengyushen/compsyn.

*The remainder of this paper is organized as follows.* Section 2 introduces the background material; Section 3 identifies the flow control variables, while Section 4 infers the predicate that enables $\vec{d}$ to be uniquely determined by a bounded sequence of $\vec{o}$; Section 5 minimizes the length of unrolled state sequence, while Section 6 characterizes the decoder's Boolean function; Sections 7 and 8 present the experimental results and related works; Finally, Section 9 sums up the conclusion.

## 2. PRELIMINARIES

### 2.1. Propositional satisfiability

The Boolean value set is denoted as $B = \{0, 1\}$. A vector of variables is represented as $\vec{v} = (v, \ldots)$. The number of variables in $\vec{v}$ is denoted as $|\vec{v}|$. If a variable $v$ is a member of $\vec{v}$, then we say $v \in \vec{v}$; otherwise we say $v \notin \vec{v}$. For a variable $v$ and a vector $\vec{v}$, if $v \notin \vec{v}$, then the new vector that contains both $v$ and all members of $\vec{v}$ is denoted as $v \cup \vec{v}$. If $v \in \vec{v}$, then the new vector that contains all members of $\vec{v}$ except $v$, is denoted as $\vec{v} - v$. For the two vectors $\vec{a}$ and $\vec{b}$, the new vector with all members of $\vec{a}$ and $\vec{b}$ is denoted as $\vec{a} \cup \vec{b}$. The set of truth valuations of $\vec{v}$ is denoted as $[\![\vec{v}]\!]$, for instance, $[\![(v_1, v_2)]\!] = \{(0,0), (0,1), (1,0), (1,1)\}$.

A Boolean formula $F$ over a variable set $V$ is constructed by connecting variables from $V$ with symbols $\neg$, $\wedge$, $\vee$ and $\Rightarrow$, which stand for logical connectives negation, conjunction, disjunction, and implication, respectively.

The propositional satisfiability problem(abbreviated as SAT) for a Boolean formula $F$ over a variable set $V$ is to find a satisfying assignment $A : V \rightarrow B$, so that $F$ can be evaluated to $1$. If $A$ exists, then $F$ is satisfiable; otherwise, it is unsatisfiable.

According to [Ganai et al. 2004], the positive and negative cofactors of $f(v_1 \ldots v \ldots v_n)$ with respect to variable $v$ are $f_{v\equiv 1} = f(v_1 \ldots 1 \ldots v_n)$ and $f_{v\equiv 0} = f(v_1 \ldots 0 \ldots v_n)$, respectively. **Cofactoring** is the action that applies 1 or 0 to $v$ to get $f_{v\equiv 1}$ or $f_{v\equiv 0}$.

Given two Boolean formulas $\phi_A$ and $\phi_B$, with $\phi_A \wedge \phi_B$ unsatisfiable, there exists a formula $\phi_I$ referring only to the common variables of $\phi_A$ and $\phi_B$ such that $\phi_A \Rightarrow \phi_I$ and $\phi_I \wedge \phi_B$ is unsatisfiable. We call $\phi_I$ the **interpolant** [Craig 1957] of $\phi_A$ with respect to $\phi_B$ and use McMillan's algorithm [McMillan 2003] to generate it.

MiniSat [Eén and Sörensson 2003] is used in this paper to solve all formulas. It generates learned clauses from conflicts, and records them to prevent the same conflict from rising again. It provides an incremental SAT mechanism that can share learned clauses between related formulas to solve them faster. This mechanism includes two procedures: $addClause(F)$ used to add a CNF formula $F$ to the clause database, and $solve(A)$ that solves $F$ with a set of literals $A$ as assumptions .

### 2.2. Finite state machine

The encoder is modeled by a finite state machine(FSM) $M = (\vec{s}, \vec{i}, \vec{o}, T)$, consisting of a state variable vector $\vec{s}$, an input variable vector $\vec{i}$, an output variable vector $\vec{o}$, and a transition function $T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \rightarrow [\![\vec{s}]\!] \times [\![\vec{o}]\!]$ that computes the next state and output variable vector from the current state and input variable vector.

The behavior of FSM $M$ can be reasoned by unrolling transition function for multiple steps. The state variable $s \in \vec{s}$, input variable $i \in \vec{i}$ and output variable $o \in \vec{o}$ at the $n$-th step are respectively denoted as $s_n$, $i_n$ and $o_n$. Furthermore, the state, the input and the output variable vectors at the $n$-th step are respectively denoted as $\vec{s}_n$, $\vec{i}_n$ and $\vec{o}_n$. A **path** is a state sequence $< \vec{s}_n, \ldots, \vec{s}_m >$ with $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$ for all $n \leq j < m$. A **loop** is a path $< \vec{s}_n, \ldots, \vec{s}_m >$ with $\vec{s}_n \equiv \vec{s}_m$.

### 2.3. The halting algorithm to determine if an input variable can be uniquely determined by a bounded sequence of output variable vector

Shen et al. [2011] proposed the first halting algorithm that iteratively unrolls the transition function. And for each iteration, it uses two approximative approaches to determine the answer. The first one is an under-approximative one that presented in 2.3.1, while the second one is an over-approximative one presented in 2.3.2. We will show in 2.3.3 that these two approaches will eventually converge to a conclusive answer.

#### *2.3.1. **The under-approximative approach**. .*

As shown in Figure 2, on the unrolled transition functions, an input variable $i \in \vec{i}$ can be uniquely determined, if there exist three integers $p$, $l$ and $r$, such that for any particular valuation of the output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, $i_{p+l}$ cannot be 0 and 1 at the same time. This is equal to the unsatisfiability of $F_{PC}(p, l, r)$ in Equation (1).

$$F_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \quad (1)$$

Here, $p$ is the length of the prefix state transition sequence. $l$ and $r$ are the lengths of the two output sequences $< \vec{o}_{p+1}, \ldots, \vec{o}_{p+l} >$ and $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ used to determine $i_{p+l}$. Line 1 of Equation (1) corresponds to the left path in Figure 2, while Line 2 corresponds to the right path in Figure 2. These two paths are of the same length. Line 3 forces these two paths' output sequences to be the same, while Line 4 forces their $i_{p+l}$ to be different. Line 5 and 6 are the assertion predicates given by the user that constrain the valid valuation on $\vec{i}$. PC in $F_{PC}$ is the abbreviation of ”parameterized complementary”, which means $F_{PC}(p, l, r)$ is used to check whether the encoder's input can be uniquely determined with the three parameters $p$, $l$ and $r$.

According to Figure 2, the first three lines of Equation (1) are two unrolled transition function sequences with the same output sequences. They can always be satisfied with the same input variable vectors and initial state vector. And the last two lines are constraints on input variable vectors. We always check their satisfiability before running our algorithm. So the unsatisfiability of $F_{PC}(p, l, r)$ always means $i_{p+l} \equiv i'_{p+l}$.

According to Equation (1), for $p' \geq p$, $l' \geq l$ and $r' \geq r$, the clause set of $F_{PC}(p', l', r')$ is a super set of $F_{PC}(p, l, r)$. So, the bounded proof of $F_{PC}(p, l, r)$'s unsatisfiability can be generalized to unbounded cases.

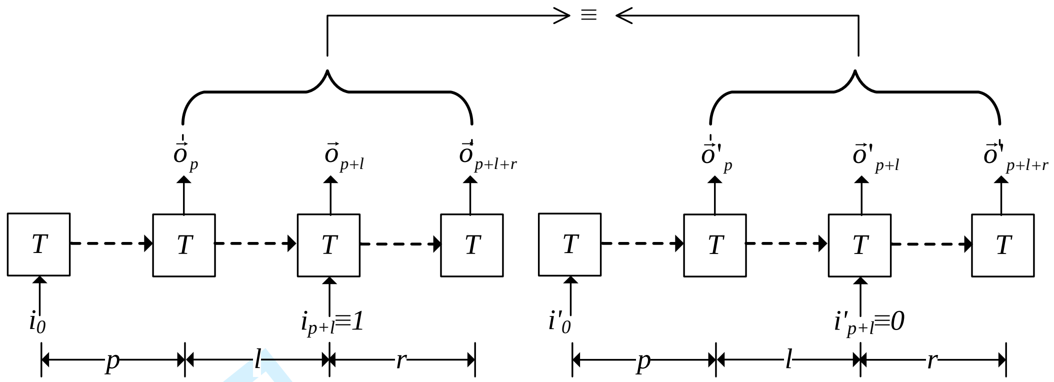


Fig. 2. The under-approximative approach checking if $i_{p+l}$ can be uniquely determined

PROPOSITION 2.1. *If $F_{PC}(p, l, r)$ is unsatisfiable, then $i_{p+l}$ can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for all larger $p$, $l$ and $r$.*

Equation (1) doesn't include an initial state, instead it uses the $p$ steps prefix state sequence $< \vec{s_0}, \ldots, \vec{s_{p-1}} >$ to propagate $assertion(\vec{i})$ into $< \vec{s_p}, \ldots, \vec{s_{p+l+r}} >$, such that some states that can't be reached with $assertion(\vec{i})$ can be eliminated. This leads to two major advantages over considering initial states: First, it simplify and speedup our algorithm by avoiding the need to compute the reachable state set or inductive invariants. Tu and Jiang [2013] rules out unreachable states by inferring inductive invariants. But it can't handle our most complex XFI benchmark [Shen et al. 2011], while our algorithms always can. Second, ignoring initial states improves the decoder's reliability by making the decoder's output depend only on bounded history of its input. Thus any corrupted $\vec{o}$ can only affect the decoder for finite number of steps.

Of course ignoring initial states has one drawback that it is a little bit too stronger than necessary. That is, it requires that $\vec{i}$ must be uniquely determined on a larger state set $R^p$ that is reachable in $p$ steps from any states, instead of on the smaller state set $R$ that is reachable from initial states. In some cases, our algorithm may fail to handle properly designed encoders. But this has never happen on all our benchmarks.

### 2.3.2. *The over-approximative approach*. .

For $F_{PC}(p, l, r)$ presented in last subsection, there are two possibilities:

(1) $i_{p+l}$ can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for some $p$, $l$ and $r$;
(2) $i_{p+l}$ can't be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for any $p$, $l$ and $r$ at all.

If it is the 1st case, then by iteratively increasing $p$, $l$ and $r$, $F_{PC}(p, l, r)$ will eventually become unsatisfiable. But if it is the 2nd case, this method will never terminate.

So, to obtain a halting algorithm, we need to distinguish these two cases. One such solution is shown in Figure 3, which is similar to Figure 2 but with three additional constraints used to detect loops on the three state sequences $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. It is formally defined in Equation (2) with the last three lines corresponding to the three new constraints used to detect loops.

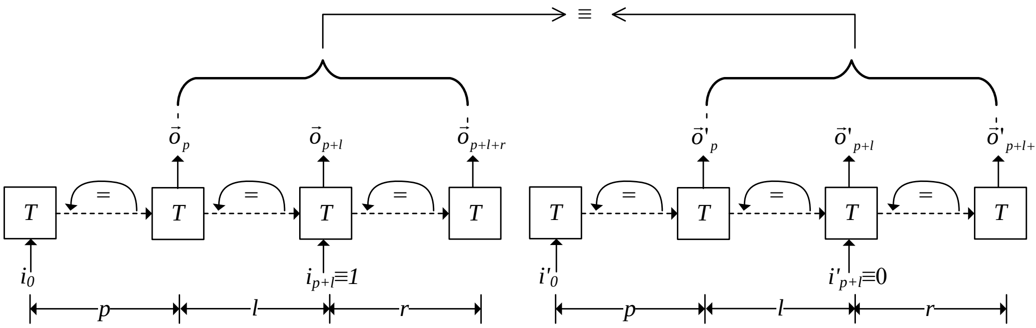


Fig. 3. The over-approximative approach checking if $i_{p+l}$ can NOT be uniquely determined

$$F_{LN}(p,l,r) := \left\{ \begin{array}{c} F_{PC}(p,l,r) \\ \wedge \quad \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge \quad \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \qquad (2)$$

LN in $F_{LN}$ stands for ”loop non-complementary”, which means $F_{LN}(p,l,r)$ with three loops is used to check whether the input variable can NOT be uniquely determined.

When $F_{LN}(p,l,r)$ is satisfiable, then $i_{p+l}$ can’t be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. More importantly, by unrolling these three loops, we can generalize the satisfiability of $F_{LN}(p,l,r)$ to all larger $p$, $l$ and $r$. This means:

PROPOSITION 2.2. *If $F_{LN}(p,l,r)$ is satisfiable, then $i_{p+l}$ cannot be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for all larger $p$, $l$ and $r$.*

#### 2.3.3. ***The full algorithm.*** .

With Propositions 2.1 and 2.2, we can generalize their bounded proof to unbounded cases. This leads to the halting Algorithm 1 that search for $p$, $l$ and $r$ that enable an input variable $i_{p+l}$ to be uniquely determined by the output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$:

(1) On the one hand, if there exists such $p$, $l$ and $r$, then let $p' := max(p,l,r)$, $l' := max(p,l,r)$ and $r' := max(p,l,r)$. From Propositions 2.1, we know that $F_{PC}(p',l',r')$ is unsatisfiable. So eventually $F_{PC}(p,l,r)$ will become unsatisfiable in Line 4;

**ALGORITHM 1:** $CheckUniqueness(i)$: The halting algorithm to determine whether $i \in \vec{i}$ can be uniquely determined by a bounded sequence of output variable vector $\vec{o}$

**Input**: The input variable $i \in \vec{i}$.
**Output**: whether $i \in \vec{i}$ can be uniquely determined by $\vec{o}$, and the value of $p$, $l$ and $r$.

```
1 p:= 0; l:= 0; r:= 0;
2 while 1 do
3     p++; l++; r++;
4     if $F_{PC}(p,l,r)$ is unsatisfiable then
5         return (1, p, l, r);
6     else if $F_{LN}(p,l,r)$ is satisfiable then
7         return (0, p, l, r);
8
```

**ALGORITHM 2:** $FindFlow(\vec{i})$:Identifying the flow control vector $\vec{f}$

**Input**: The input variable vector $\vec{i}$.
**Output**: $\vec{f} \subset \vec{i}$ and $\vec{d} \subset \vec{i}$ that can and can't be uniquely determined ,and the maximal $p$, $l$ and $r$ reached in this searching.

```
1  $\vec{f} := \{\}$;$\vec{d} := \{\}$;$p$:= 0 ; $l$:= 0 ; $r$:= 0 ;
2  while $\vec{i} \neq \{\}$ do
3      assume $i \in \vec{i}$;
4      $p++;\;\; l++;\;\; r++;$
5      if $F_{PC}(p,l,r)$ is unsatisfiable for $i$ then
6          $\vec{f} := i \cup \vec{f}$;
7          $\vec{i} := \vec{i} - i$;
8      else if $F_{LN}(p,l,r)$ is satisfiable for $i$ and assume its satisfying assignment is $A$ then
9          foreach $j \in \vec{i}$ do
10             if $A(j_{p+l}) \neq A(j'_{p+l})$ then
11                 $\vec{i} := \vec{i} - j$;
12                 $\vec{d} := j \cup \vec{d}$
13
14 return ($\vec{f}$, $p$, $l$, $r$)
```

(2) On the other hand, if there doesn't exist such $p$, $l$ and $r$, then eventually $p$, $l$ and $r$ will be larger than the encoder's longest path without loop, which means that there will be three loops in $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. This will make $F_{LN}(p,l,r)$ satisfiable in Line 6.

Both cases will lead to this Algorithm's termination.

## 3. IDENTIFYING FLOW CONTROL VECTOR

We will first introduce how to find out the flow control vector $\vec{f}$ in Subsection 3.1, and then introduce how to speed up it with incremental SAT in Subsection 3.2.

### 3.1. Finding out flow control vector $\vec{f}$

To facilitate the presentation of our algorithm, we assume that the input variable vector $\vec{i}$ can be partitioned into the flow control vector $\vec{f}$ and the data vector $\vec{d}$.

The flow control vector $\vec{f}$ is used to indicate the validness of $\vec{d}$. So, for a properly designed encoder, $\vec{f}$ should always be uniquely determined by a bounded sequence of the encoder's output vector $\vec{o}$, or else the decoder cannot recognize the validness of $\vec{d}$.

Thus, Algorithm 2 is proposed to identify $\vec{f}$. At Line 2, a while loop is used to iterate on all $i \in \vec{i}$. At Line 6, the input variable $i$ that can be uniquely determined will be added to vector $\vec{f}$. On the other hand, when $\vec{i}$ is very long, the run time overhead of testing each $i \in \vec{i}$ one by one would also be very large. To speed up this testing procedure, when $F_{LN}(p,l,r)$ is satisfiable at Line 8, every $j \in \vec{i}$ that has different values for $j_{p+l}$ and $j'_{p+l}$ in the satisfying assignment of $F_{LN}(p,l,r)$ can also be moved to $\vec{d}$ at Line 10, because their own $F_{LN}(p,l,r)$ is also satisfiable.

In some particular case, some data variable $d \in \vec{d}$ can be uniquely determined, just like a flow control variable $f \in \vec{f}$. In this case, $d$ may be identified as a flow control variable by Algorithm 2. But this doesn't harm our overall framework, because the decoder's Boolean function of $d$ can still be correctly characterized in Section 6.

### 3.2. Speeding up with incremental SAT

We can further speedup Algorithm 2 by employing the MiniSat's incremental SAT mechanism mentioned in Subsection 2.1. $F_{PC}(p,l,r)$ in Equation (1) can be partitioned into the following two equations by moving the third line of it into Equation (4):

$$C_{PC}(p,l,r) := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3}$$

$$A_{PC}(p,l,r) := \{\ i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0\ \} \tag{4}$$

Similarly we can partition $F_{LN}(p,l,r)$ in Equation (2) into two equations:

$$C_{LN}(p,l,r) := \left\{ \begin{array}{lc} & C_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{5}$$

$$A_{LN}(p,l,r) := \{\ i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0\ \} \tag{6}$$

Obviously $C_{PC}$ and $C_{LN}$ are independent of any particular $i \in \vec{i}$, so they can be added into the clause database of MiniSat solver by calling $addClause(C_{PC})$ or

**ALGORITHM 3:** $FindFlowIncSAT(\vec{i})$:Identifying the flow control vector with incremental SAT

**Input**: The input variable vector $\vec{i}$.

**Output**: $\vec{f} \subset \vec{i}$ and $\vec{d} \subset \vec{i}$ that can and can't be uniquely determined, and the maximal $p$, $l$ and $r$ reached in this searching.

```
$\vec{f} := \{\};\vec{d} := \{\}; p := 0 ; l := 0 ; r := 0$ ;
while $\vec{i} \neq \{\}$ do
    $p++;\ \ l++;\ \ r++;$
    $addClause(C_{PC}(p,l,r))$;
    foreach $i \in \vec{i}$ do
        if $solve(A_{PC}(p,l,r))$ is unsatisfiable for $i$ then
            $\vec{i} := \vec{i} - i$;
            $\vec{f} := i \cup \vec{f}$;
    $addClause(C_{LN}(p,l,r))$;
    foreach $i \in \vec{i}$ do
        if $solve(A_{LN}(p,l,r))$ is satisfiable for $i$ and satisfying assignment is $A$ then
            foreach $j \in \vec{i}$ do
                if $A(j_{p+l}) \neq A(j'_{p+l})$ then
                    $\vec{i} := \vec{i} - j$;
                    $\vec{d} := j \cup \vec{d}$
return $(\vec{f}, p, l, r)$
```

$addClause(C_{LN})$. At the same time, all clauses in $A_{PC}$ and $A_{LN}$ contain only one literal, so they can be used as the assumptions in calling $solve$ procedure of MiniSat solver.

Thus, with these new equations, we can change Algorithm 2 to Algorithm 3 with incremental SAT. The major changes are the two new $addClause$ in Line 4 and 9, and the two new $solve$ in Line 6 and 11. They are the procedures provided by MiniSat's incremental SAT mechanism mentioned in Subsection 2.1.

## 4. INFERRING PREDICATE THAT ENABLES THE ENCODER'S DATA VECTOR TO BE UNIQUELY DETERMINED

In Subsection 4.1, we propose an algorithm to characterize a Boolean function that makes a Boolean formula satisfiable. In Subsection 4.2, we apply this algorithm to infer $valid(\vec{f})$, the predicate that enables $\vec{d}$ to be uniquely determined.

### 4.1. Characterizing a function that makes a Boolean formula satisfiable

For a particular Boolean relation $R(\vec{a}, \vec{b}, t)$, we have the following two assumptions:

(1) **Assumption 1** : $R(\vec{a}, \vec{b}, 0) \wedge R(\vec{a}, \vec{b}, 1)$ is unsatisfiable. $\vec{a}$ and $\vec{b}$ are respectively called the important and the non-important variable vectors, $t$ is the target variable.

(2) **Assumption 2** : $R(\vec{a}, \vec{b}, t)$ is satisfiable for all valuations of $\vec{a}$.

In the remainder of this paper, when we use the algorithm introduced in this subsection, we will show that these two assumptions are fulfilled.

We need to characterize a Boolean function $FSAT_R(\vec{a})$, which covers and only covers all the valuations of $\vec{a}$ that can make $R(\vec{a}, \vec{b}, 1)$ satisfiable. It is formally defined below:

$$FSAT_R(\vec{a}) := \begin{cases} 1 & \exists \vec{b}.R(\vec{a}, \vec{b}, 1) \\ 0 & otherwise \end{cases} \tag{7}$$

Thus, a naive algorithm of computing $FSAT_R(\vec{a})$ is to enumerate all valuations of $\vec{a}$, and collect all those valuations that make $R(\vec{a}, \vec{b}, 1)$ satisfiable. But the number of valuations to be enumerated is $2^{|\vec{a}|}$, which will prevent this algorithm from terminating within reasonable time for a large $\vec{a}$.

We can speed up this naive algorithm by expanding each valuation of $\vec{a}$ to a larger set with cofactoring [Ganai et al. 2004] and Craig interpolant [McMillan 2003]. Intuitively, assume that $R(\vec{a}, \vec{b}, 1)$ is satisfiable with a satisfying assignment $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0, 1\}$, the following new formula can be constructed by cofactoring [Ganai et al. 2004]:

$$R(\vec{a}, A(\vec{b}), 1) := R(\vec{a}, \vec{b}, 1)_{b \equiv A(b)} \tag{8}$$

Because $R(\vec{a}, A(\vec{b}), 0) \wedge R(\vec{a}, A(\vec{b}), 1)$ is unsatisfiable, the interpolant $ITP(\vec{a})$ of $R(\vec{a}, A(\vec{b}), 1)$ with respect to $R(\vec{a}, A(\vec{b}), 0)$ can be used as an over-approximation of the set of $\vec{a}$ that makes $R(\vec{a}, A(\vec{b}), 1)$ satisfiable. At the same time, $ITP(\vec{a}) \wedge R(\vec{a}, A(\vec{b}), 0)$ is unsatisfiable, so $ITP(\vec{a})$ covers nothing that can make $R(\vec{a}, A(\vec{b}), 0)$ satisfiable. Thus, $ITP(\vec{a})$ covers exactly the set of valuations of $\vec{a}$ that can make $R(\vec{a}, A(\vec{b}), 1)$ satisfiable.

Based on the foregoing discussion, Algorithm 4 is proposed to characterize $FSAT_R(\vec{a})$ in Equation (7). Line 2 checks whether there is still some new valuation of $\vec{a}$ that can make $R(\vec{a}, \vec{b}, 1)$ satisfiable, but hasn't been covered by $FSAT_R(\vec{a})$. Lines 4 and 5 assign the value of $\vec{b}$ from the satisfying assignment to $R(\vec{a}, \vec{b}, 1)$ and $R(\vec{a}, \vec{b}, 0)$ respectively. This will make $\vec{b}$ to no longer appear as free variables in these two formulas.

Thus, $\phi_A \wedge \phi_B$ in Line 6 is unsatisfiable, and the common variables of $\phi_A$ and $\phi_B$ is $\vec{a}$. So an interpolant $ITP(\vec{a})$ can be generated with McMillan's algorithm [McMillan 2003]. $ITP(\vec{a})$ is added to $FSAT_R(\vec{a})$ in Line 7 and ruled out in Line 2 again.

Each iteration of the while loop in Algorithm 4 adds at least a valuation of $\vec{a}$ to $FSAT_R(\vec{a})$, which means that $FSAT_R(\vec{a})$ is a Boolean function that covers a bounded and strictly increasing set of valuations of $\vec{a}$. So Algorithm 4 is a halting one.

### 4.2. Inferring $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined

As shown in Figure 4, we first define $\neg FSAT_{PC}(p,l,r)$, a monotonically growing under-approximation of $valid(\vec{f})$ in 4.2.1. And then we define $\neg FSAT_{LN}(p,l,r)$, a monotonically shrinking over-approximation of $valid(\vec{f})$ in 4.2.2. And finally we show that they will converge to $valid(\vec{f})$ in 4.2.3. Its correctness will be proved in Subsection 4.3.

*4.2.1. **Computing monotonically growing under-approximation of** $valid(\vec{f})$. .*
By replacing $i$ in Equation (1) with $\vec{d}$, we have:

$$F^d_{PC}(p,l,r) := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{9}$$

Here, $\vec{d}_{p+l} \neq \vec{d'}_{p+l}$ means some bit in $\vec{d}_{p+l}$ isn't equal to the corresponding bit in $\vec{d'}_{p+l}$. If $F^d_{PC}(p,l,r)$ is satisfiable, then $\vec{d}_{p+l}$ cannot be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. We define $T_{PC}(p,l,r)$ by collecting the 3rd line of Equation (9):

$$T_{PC}(p,l,r) := \left\{ \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \right\} \tag{10}$$

**ALGORITHM 4:** $CharacterizingFormulaSAT(R,\vec{a},\vec{b},t)$:Characterizing a Boolean function over $\vec{a}$ that can make $R(\vec{a},\vec{b},1)$ satisfiable

```
Input: The Boolean formula $R(\vec{a},\vec{b},t)$, its important variable vector $\vec{a}$, its non-important
       variable vector $\vec{b}$, and its target variable $t$.
Output: $FSAT_R(\vec{a})$ that makes $R(\vec{a},\vec{b},1)$ satisfiable.
1 $FSAT_R(\vec{a}) := 0$ ;
2 while $R(\vec{a},\vec{b},1) \wedge \neg FSAT_R(\vec{a})$ is satisfiable do
3     assume $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0,1\}$ is the satisfying assignment ;
4     $\phi_A(\vec{a}) := R(\vec{a},A(\vec{b}),1)$ ;
5     $\phi_B(\vec{a}) := R(\vec{a},A(\vec{b}),0)$ ;
6     assume $ITP(\vec{a})$ is the Craig interpolant of $\phi_A$ with respect to $\phi_B$ ;
7     $FSAT_R(\vec{a}) := ITP(\vec{a}) \vee FSAT_R(\vec{a})$ ;
8 return $FSAT_R(\vec{a})$
```

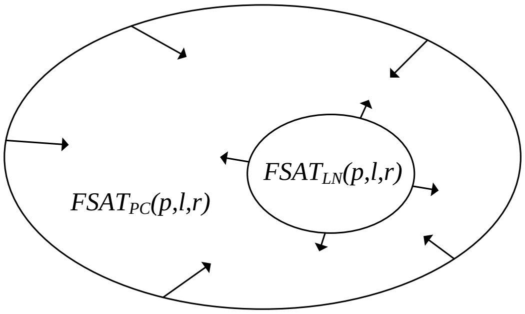


Fig. 4. The monotonicity of $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$

By substituting $T_{PC}(p,l,r)$ back into $F^{d}_{PC}(p,l,r)$, we have a new formula:

$$F'^{d}_{PC}(p,l,r,t) := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{11}$$

Obviously $F^{d}_{PC}(p,l,r)$ and $F'^{d}_{PC}(p,l,r,1)$ are equivalent. We further define:

$$\vec{a} := \vec{f}_{p+l} \tag{12}$$

$$\vec{b} := \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0 \leq x \leq p+l+r, x \neq (p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{13}$$

Thus, $\vec{a} \cup \vec{b}$ is the vector that contains all the input variable vectors $< \vec{i}_0, \ldots, \vec{i}_{p+l+r} >$ and $< \vec{i'}_0, \ldots, \vec{i'}_{p+l+r} >$ at all steps for the two sequences of unrolled transition function. It also contains the two initial states $\vec{s}_0$ and $\vec{s'}_0$. In addition, the transition function $T$ in the first two lines of Equation (11) is a function that computes the next state and the output variable vector from the current state and input variable vector. So $\vec{a}$ and $\vec{b}$ can uniquely determine the value of $t$ in $F'^{d}_{PC}(p,l,r,t)$.

That is, with $\vec{a}$ defined in (12), $\vec{b}$ defined in (13) and $F'^{d}_{PC}(p,l,r,t)$ as $R(\vec{a},\vec{b},t)$, Assumption 1 and 2 in Subsection 4.1 are both fulfilled. Thus, for a particular combination of $p$, $l$ and $r$, the Boolean function over $\vec{f}_{p+l}$ that makes $F'^{d}_{PC}(p,l,r,1)$ satisfiable can be computed by calling Algorithm 4 with $F'^{d}_{PC}(p,l,r,t)$, $\vec{a}$ and $\vec{b}$ defined above:

$$FSAT_{PC}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{PC}(p,l,r,t),\vec{a},\vec{b},t) \tag{14}$$

So $FSAT_{PC}(p,l,r)$ is the set of $\vec{f}_{p+l}$ that makes $F^{d}_{PC}(p,l,r)$ satisfiable. Thus, its negation $\neg FSAT_{PC}(p,l,r)$ is the set of $\vec{f}_{p+l}$ that makes $F^{d}_{PC}(p,l,r)$ unsatisfiable.

Again according to Proposition 2.1, the unsatisfiable proof of $F^{d}_{PC}(p,l,r)$ can be generalized to all larger $p$, $l$ and $r$. So every valuation of $\vec{f}$ covered by $\neg FSAT_{PC}(p,l,r)$ can also make $F^{d}_{PC}(p,l,r)$ unsatisfiable for all larger $p$, $l$ and $r$. So we have:

PROPOSITION 4.1. *$\neg FSAT_{PC}(p,l,r)$ is an under-approximation of $valid(\vec{f})$ that grows monotonically with respect to $p$, $l$ and $r$.*

This is shown intuitively in Figure 4..

*4.2.2.* ***Computing monotonically shrinking over-approximation of*** $valid(\vec{f})$. . Similarly, by replacing $i$ in $F_{LN}(p,l,r)$ of Equation (2) with $\vec{d}$, we have:

$$F^d_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{15}$$

If $F^d_{LN}(p,l,r)$ is satisfiable, then $\vec{d}_{p+l}$ cannot be uniquely determined by $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$. Furthermore, similar to Proposition 2.2, by unrolling those three loops in the last three lines of Equation (15), we can prove that $\vec{d}_{p+l}$ cannot be uniquely determined for any larger $p$, $l$ and $r$. We further define a new formula $T_{LN}(p,l,r)$ by collecting the 3rd line and the last three lines of Equation (15):

$$T_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{16}$$

By replacing the 3rd line and the last three lines of Equation (15) with $T_{LN}(p,l,r)$, we got:

$$F'^d_{LN}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{17}$$

Obviously $F^d_{LN}(p,l,r)$ and $F'^d_{LN}(p,l,r,1)$ are equivalent. Thus, with $\vec{a}$ defined in (12), $\vec{b}$ defined in (13) and $F'^d_{LN}(p,l,r,1)$ as $R(\vec{a},\vec{b},t)$, Assumption 1 and 2 in Subsection 4.1 are both fulfilled. Thus, for a particular valuation of $p$, $l$ and $r$, the function over $\vec{f}_{p+l}$ that makes $F^d_{LN}(p,l,r)$ satisfiable can be computed by:

$$FSAT_{LN}(p,l,r) := CharacterizingFormulaSAT(F'^d_{LN}(p,l,r,t),\vec{a},\vec{b},t) \tag{18}$$

Again according to Proposition 2.2, the satisfiable proof of $F^d_{LN}(p,l,r)$ can be generalized to all larger $p$, $l$ and $r$. So every valuation of $\vec{f}$ covered by $FSAT_{LN}(p,l,r)$

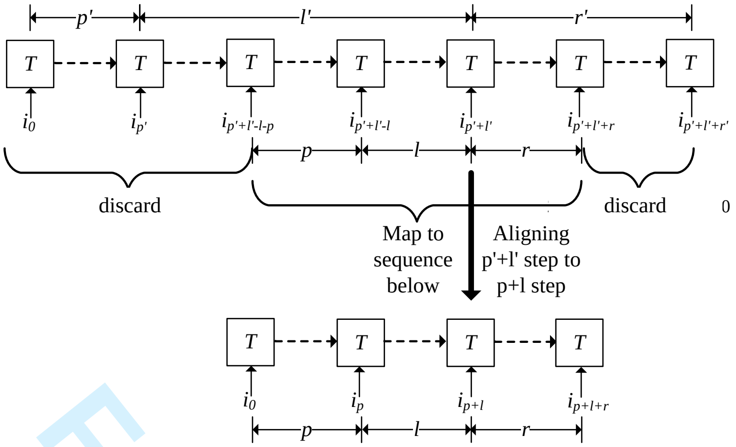


Fig. 5. Mapping $F'^{d}_{PC}(p',l',r',1)$ to $F'^{d}_{PC}(p,l,r,1)$ by aligning $(p'+l')$-step to $(p+l)$-step.

can also make $F^{d}_{LN}(p,l,r)$ satisfiable for all larger $p$, $l$ and $r$. So $FSAT_{LN}(p,l,r)$ grow monotonically, and is a subset of $\neg valid(\vec{f})$. Thus we have the following proposition:

PROPOSITION 4.2. *$\neg FSAT_{LN}(p,l,r)$ is an over-approximation of $valid(\vec{f})$ that shrinks monotonically.*

This is shown intuitively in Figure 4.

### 4.2.3. *The algorithm to compute* $valid(\vec{f})$. .

With Propositions 4.1 and 4.2, the algorithm that infers the predicate $valid(\vec{f}_{p+l})$ is shown in Algorithm 5. It just iteratively increases the value of $p$, $l$ and $r$, until $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ is unsatisfiable, which means that $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$ converge. In this case, $\neg FSAT_{PC}(p,l,r)$ is return as $valid(\vec{f})$.

The proofs of its termination and correctness are given in the next subsection.

## 4.3. Proofs of termination and correctness

First we need to prove the following three lemmas:

LEMMA 4.3. $\mathbf{FSAT_{PC}(p,l,r)}$ *in Algorithm 5 monotonically decreases with respect to $p$, $l$ and $r$.*

PROOF. For any $p' > p$,$l' > l$ and $r' > l$, assume $A : \vec{f}_{p'+l'} \rightarrow B$ is a valuation of the flow control vector at $(p'+l')$-step. Further assume that $A$ is covered by $FSAT_{PC}(p',l',r')$. According to Equation (14) and Algorithm 4, $A$ can make $F'^{d}_{PC}(p',l',r',1)$ satisfiable. Assume this satisfying assignment of $F'^{d}_{PC}(p',l',r',1)$ is $A'$. Then $A(\vec{f}_{p'+l'}) \equiv A'(\vec{f}_{p'+l'})$.

**ALGORITHM 5:** $InferringUniqueFormula$: inferring the predicate $valid(\vec{f}_{p+l})$ that enables $\vec{d}_{p+l}$ to be uniquely determined

```
$p$:= $p_{max}$; $l$:= $l_{max}$; $r$:= $r_{max}$ ;
while $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ is satisfiable do
    $p$ ++ ; $l$ ++ ; $r$ ++ ;
return $\neg FSAT_{LN}(p,l,r)$
```

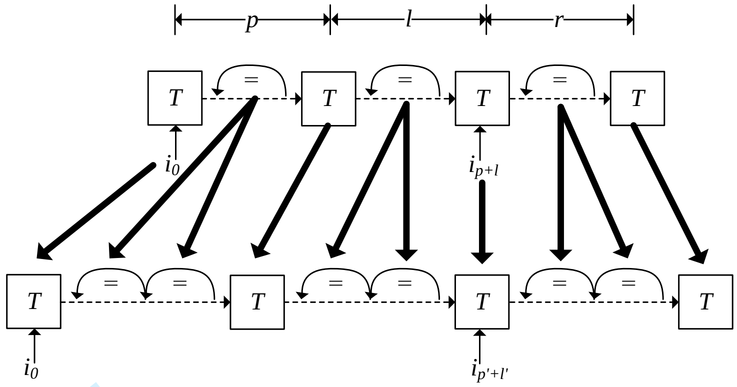


Fig. 6. Mapping $F'^d_{LN}(p, l, r, 1)$ to $F'^d_{LN}(p', l', r', 1)$ by aligning $(p + l)$-step to $(p' + l')$-step and unrolling the three loops.

As shown in Figure 5, we can map the state, input and output vectors in $F'^d_{PC}(p', l', r', 1)$ to that of $F'^d_{PC}(p, l, r, 1)$ by aligning the $(p' + l')$-step to $(p + l)$-step, and discard the two prefix and postfix state transition sequences. Formally, for each $p'+l'-l-p \le n \le p'+l'+r$, we map $s_n$ in $F'^d_{PC}(p', l', r', 1)$ to $s_{n-p'-l'+l+p}$ in $F'^d_{PC}(p, l, r, 1)$. $i_n$ and $o_n$ are also mapped similarly.

With this mapping, we can transform the satisfying assignment $A'$ of $F'^d_{PC}(p', l', r', 1)$ to yet another satisfying assignment $A''$ of $F'^d_{PC}(p, l, r, 1)$. By restricting the domain of $A''$ to $\vec{f}_{p+l}$, we got the fourth satisfying assignment $A''' : \vec{f}_{p+l} \rightarrow B$. According to the mapping presented above, we know that $A''' \equiv A$. Thus, every $A$ covered by $FSAT_{PC}(p', l', r')$ is also covered by $FSAT_{PC}(p, l, r)$.

Thus, $FSAT_{PC}(p, l, r)$ monotonically decreases with respect to $p$, $l$ and $r$. □

LEMMA 4.4. $\mathbf{FSAT_{LN}(p, l, r)}$ *in Algorithm 5 monotonically increases with respect to $p$, $l$ and $r$.*

PROOF. For any $p' > p$,$l' > l$ and $r' > l$, assume $A : \vec{f}_{p+l} \rightarrow B$ is a Boolean valuation of the flow control vector at $(p + l)$-step. Further assume that $A$ is covered by $FSAT_{LN}(p, l, r)$. So $A$ can make $F'^d_{LN}(p, l, r, 1)$ satisfiable. Assume this satisfying assignment of $F'^d_{LN}(p, l, r, 1)$ is $A'$. We know that $A(\vec{f}_{p+l}) \equiv A'(\vec{f}_{p+l})$.

As shown in Figure 6, we can map $F'^d_{LN}(p, l, r, 1)$ to $F'^d_{LN}(p', l', r', 1)$ by aligning $(p+l)$-step to $(p' + l')$-step, and unrolling those three loops. In this way we can get another satisfying assignment $A''$ of $F'^d_{LN}(p', l', r', 1)$.

By restricting the domain of $A''$ to $\vec{f}_{p'+l'}$, we can get the fourth satisfying assignment $A''' : \vec{f}_{p'+l'} \rightarrow B$. It is obvious that $A \equiv A'''$. This means that every valuation covered by $FSAT_{LN}(p, l, r)$ is also covered by $FSAT_{LN}(p', l', r')$.

So $FSAT_{LN}(p, l, r)$ grow monotonically with respect to $p$, $l$ and $r$. □

LEMMA 4.5. $FSAT_{LN}(p, l, r) \Rightarrow FSAT_{PC}(p, l, r)$

PROOF. It is obvious that the clause set of $F'^d_{LN}(p, l, r, 1)$ is a super set of $F'^d_{PC}(p, l, r, 1)$. So every satisfying assignment of $F'^d_{LN}(p, l, r, 1)$ can also satisfy $F'^d_{PC}(p, l, r, 1)$. so $FSAT_{LN}(p, l, r) \Rightarrow FSAT_{PC}(p, l, r)$ holds. □

These three lemmas are depicted intuitively in Figure 4, which makes it obvious that $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ monotonically decreases in Algorithm 5. With these lemmas, let's first prove that Algorithm 5 is a halting one.

THEOREM 4.6. *Algorithm 5 is a halting algorithm.*

PROOF. As the encoder is represented by a finite state machine, the length of the longest path without loop is finite. If Algorithm 5 doesn't halt, then eventually the values of $p$, $l$ and $r$ in Algorithm 5 will be larger than the length of the longest path without loop, which means there will be loops in these three state sequences $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ and $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$. Thus, every satisfying assignment of $F'^{d}_{PC}(p,l,r,1)$ also satisfies $F'^{d}_{LN}(p,l,r,1)$, which means $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ is unsatisfiable. This will lead to the termination of Algorithm 5. So, it is a halting algorithm. □

We will then prove the correctness of Algorithm 5.

THEOREM 4.7. *$\neg FSAT_{LN}(p,l,r)$ returned by Algorithm 5 covers and only covers all valuations of $\vec{f}$ that enable $\vec{d}$ to be uniquely determined by a bounded sequence of $\vec{o}$.*

PROOF. First prove the covering case. $FSAT_{LN}(p,l,r)$ covers a set of $\vec{f}$ that make $\vec{d}$ to be not uniquely determined for some particular $p$, $l$ and $r$. So $\neg FSAT_{LN}(p,l,r)$ rules them out and covers all valuations of $\vec{f}$ that enable $\vec{d}$ to be uniquely determined.

Then prove the only covering case by contradiction. If $A$ is a valuation of $\vec{f}$ covered by $\neg FSAT_{LN}(p,l,r)$ that makes $\vec{d}$ to be **NOT** uniquely determined for some particular $p'$, $l'$ and $r'$, then we have:

(1) $FSAT_{LN}(p',l',r')$ also covers $A$. Then according to Lemma 4.4, for all $p'' > max(p',p)$,$l'' > max(l',l)$ and $r'' > max(r',r)$, $FSAT_{LN}(p'',l'',r'')$ also covers $A$.

(2) $FSAT_{LN}(p,l,r)$ doesn't cover $A$. $FSAT_{LN}(p,l,r)$ is the last one computed by Algorithm 5 before it terminates. Which means that the $valid(\vec{f})$'s under-approximation and over-approximation have converge. So for all $p'' > max(p',p)$,$l'' > max(l',l)$ and $r'' > max(r',r)$, $FSAT_{LN}(p'',l'',r'')$ must be equal to $FSAT_{LN}(p,l,r)$. Thus $FSAT_{LN}(p'',l'',r'')$ also doesn't cover $A$.

These two cases conflict with each other. So there is no such $A$ covered by $\neg FSAT_{LN}(p,l,r)$ that makes $\vec{d}$ to be **NOT** uniquely determined for some particular $p'$, $l'$ and $r'$. So the only covering case is proved. □

## 5. MINIMIZING THE LENGTH OF UNROLLED TRANSITION FUNCTION

We first introduce why and how to remove the redundancy of $l$ and $r$ in Subsection 5.1, and then present another possible structure of our algorithm in Subsection 5.2, and discuss why we have chosen the one in Subsection 5.1 instead of Subsection 5.2.

### 5.1. Minimizing the valuation of $l$ and $r$

As Algorithm 5 increases $p$, $l$ and $r$ concurrently, there may be some redundancy leading to unnecessary overhead in the decoder's area and delay. According to Figure 2, $r$ affects both the decoder's delay and area, $l$ affects only the decoder's area, while $p$ doesn't affect the decoder.

So as shown in Algorithm 6 we chose to first minimize $r$, then minimize $l$. To simplify the presentation, we will only introduce the $r$ case. In Line 2, when $F_{PC}(p,l,r'-1) \wedge valid(\vec{f}_{p+l})$ is satisfiable, then $r'$ is the last one that makes it unsatisfiable, we

**ALGORITHM 6:** $RemoveRedundancy(p, l, r)$

```
1  for $r' := r \to 1$ do
2      if $F_{PC}(p, l, r' - 1) \wedge valid(\vec{f}_{p+l})$ is satisfiable then
3          break
4      else if $r' \equiv 1$ then
5          $r' := r' - 1$
6          break
7  for $l' := l \to 1$ do
8      if $F_{PC}(p, l' - 1, r') \wedge valid(\vec{f}_{p+l'-1})$ is satisfiable then
9          break
10     else if $l' \equiv 1$ then
11         $l' := l' - 1$
12         break
13 return $< l', r' >$
```

return it directly. On the other hand, if $r' \equiv 0$ can still make $F_{PC}(p, l, r') \wedge valid(\vec{f}_{p+l})$ unsatisfiable at Line 4, we directly return 0.

### 5.2. An alternative approach for comparison

In above discussion, we first find out the flow control vector by increasing $p$, $l$ and $r$ concurrently in Algorithm 3, and then minimize their valuations with Algorithm 6. This approach need to call SAT solver for $O(n)$ times, with $n = max(p, l, r)$.

There is another possible way to do this job in Algorithm 3, that is, increasing $p$, $l$ and $r$ one by one with three nested loops, instead of concurrently. This approach need to call SAT solver for $O(n^3)$ times,

We will show in Subsection 7.6 that, increasing $p$, $l$ and $r$ concurrently and then minimized them with Algorithm 6 is much faster than increasing them separately.

## 6. CHARACTERIZING THE DECODER'S BOOLEAN FUNCTION

The encoder's input vector $\vec{i}$ has been partitioned into the flow control vector $\vec{f}$ and data vector $\vec{d}$. The algorithms to characterize their Boolean functions are different, so they are discussed separately in the following two subsections.

### 6.1. Characterizing the decoder's Boolean function that computes $\vec{f}$

Each variable $f \in \vec{f}$ can be uniquely determined by a bounded sequence of the encoder's output. So, for each particular valuation of the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, $f_{p+l}$ cannot be 0 and 1 at the same time. Thus, the decoder's Boolean function that computes $f_{p+l}$ is exactly the Craig interpolant of $\phi_A$ with respect to $\phi_B$:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & f_{p+l} \equiv 1 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{19}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & f'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{20}$$

It is obvious that $\phi_A \wedge \phi_B$ equals $F_{PC}(p,l,r)$ in Equation (1), so it is unsatisfiable. The common variable set of $\phi_A$ and $\phi_B$ is $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. So, a Craig interpolant $ITP$ can be derived by McMillan's algorithm [McMillan 2003] from the unsatisfiability proof of $\phi_A \wedge \phi_B$, which covers all values of $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ that make $f_{p+l} \equiv 1$. At the same time, $ITP \wedge \phi_B$ is unsatisfiable, so $ITP$ covers nothing that can make $f_{p+l} \equiv 0$. Thus, $ITP$ is the decoder's Boolean function that computes $f \in \vec{f}$.

To speedup characterizing the decoder's Boolean function for all $f \in \vec{f}$, we can employ the incremental SAT mechanism in MiniSat again by:

(1) Removing $f_{p+l} \equiv 1$ from $\phi_A$ and $f'_{p+l} \equiv 0$ from $\phi_B$.
(2) Adding $\phi_A \wedge \phi_B$ into the MiniSat's clause database.
(3) Solving with assumptions $f_{p+l} \equiv 1$ and $f'_{p+l} \equiv 0$ for each $f \in \vec{f}$. And generating Craig interpolants separately.

### 6.2. Characterizing the decoder's Boolean function that computes $\vec{d}$

Assume that the predicate over $\vec{f}$ inferred by Algorithm 5 is $valid(\vec{f})$. Let's define the following two formulas for each data variable $d \in \vec{d}$:

$$\phi'_A := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & d_{p+l} \equiv 1 \\ \wedge & valid(\vec{f}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{21}$$

$$\phi'_B := \left\{ \begin{array}{lc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & d'_{p+l} \equiv 0 \\ \wedge & valid(\vec{f'}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{22}$$

Each $d \in \vec{d}$ can be uniquely determined only when $valid(\vec{f})$ holds. So, $\phi'_A \wedge \phi'_B$ is unsatisfiable. A Craig interpolant $ITP$ can be derived by McMillan's algorithm [McMillan 2003], and used as the decoder's Boolean function that computes $d \in \vec{d}$.

Furthermore, when $valid(\vec{f}_{p+l})$ doesn't hold, the data variable $d \in \vec{d}_{p+l}$ cannot be uniquely determined. So, no function can be used to calculate its value. But this isn't a problem, because the decoder is supposed to only recover the value of control flow vector $\vec{f}$, and ignore the exact value of $\vec{d}$ in this case.

Similarly, we can also use the incremental SAT approach in Subsection 6.1 to speedup characterizing the decoder's Boolean function for all $d \in \vec{d}$.

## 7. EXPERIMENTAL RESULTS

We have implemented these algorithms in OCaml language, and solved the generated CNF formulas with MiniSat 1.14 [Eén and Sörensson 2003]. All experiments have been run on a server with 16 Intel Xeon E5648 processors at 2.67GHz, 192GB memory, and CentOS 5.4 Linux. All these experimental results and programs can be downloaded from https://github.com/shengyushen/compsyn.

### 7.1. Benchmarks

Table I shows all benchmarks used in this paper. They come from our previous paper [Shen et al. 2012] and [Liu et al. 2012]. Each column of Table I shows respectively the

Table I. Benchmarks

| | Names | # in/out | # reg/gate | Circuit area | Description of Encoders | How to handle |
|---|---|---|---|---|---|---|
| Benchmarks in Shen [2012] with flow control | PCIE2 | 10 / 11 | 22 / 149 | 326 | PCIE 2.0 [PCI-SIG 2009] | Subsection 7.2 |
| | XGXS | 10 / 10 | 17 / 249 | 572 | 10Gb Ethernet in clause 48 of [IEEE 2012] | Subsection 7.3 |
| | T2Eth | 14 / 14 | 53 / 427 | 947 | Ethernet encoder of UltraSPARC T2 processor | Subsection 7.4 |
| Benchmarks in Shen [2012] without flow control | XFI | 72 / 66 | 72 / 5086 | 12381 | 10Gb Ethernet in clause 49 of [IEEE 2012] | Comparing our algorithm to Liu [2012] in Subsection 7.5 |
| | SCRAMBLER | 64/64 | 58/353 | 1034 | Making a bit stream with lots of 01 flipping | |
| Benchmarks in Liu [2012] | CC_3 | 1/3 | 6/22 | 54 | convolution code with length 3 | |
| | CC_4 | 1/3 | 7/26 | 63 | convolution code with length 4 | |
| | HM(7,4) | 4/7 | 3/38 | 103 | Hamming encoder with 4 inputs and 7 outputs | |
| | HM(15 ,11) | 11/15 | 4/102 | 317 | Hamming encoder with 11 inputs and 15 outputs | |

number of inputs, outputs, registers and gates of each benchmarks. The area column shows the area of the encoder when mapped to mcnc.genlib library by ABC [Berkeley Logic Synthesis and Verification Group 2008] with script ”strash; dsd; strash; dc2; dc2; dch; map”. In the remainder of this paper, all areas, gate number and delay are obtained in the same setting such that we can compare them to that of [Liu et al. 2012].

The last column of Table I shows how will we present their experimental result:

(1) By studying the 5 benchmarks used in our previous papers [Shen et al. 2012], we found that most of them have built-in flow control mechanisms. This isn't a surprise to us, because these benchmarks all come from real industrial projects. These three benchmarks will be presented in Subsection 7.2,7.3 and 7.4.
(2) For all other benchmarks in Table I without flow control mechanism, if their input variables can always uniquely determined, our algorithm can recognize all their input variables as flow control variables, and directly generate their decoder's Boolean functions. Their experimental results will be presented in Subsection 7.5.

In Subsection 7.6, we will compare run time overhead of two different approaches:

(1) Increasing $p$, $l$ and $r$ concurrently in Algorithm 5, then minimizing in Algorithm 6.
(2) Increasing $p$, $l$ and $r$ separately with three nested loops in Algorithm 5.

In Subsection 7.7, we will compare run times, areas and delay between the two approaches that do and do not minimize $p$, $l$ and $r$ by Algorithm 6.

Finally in Subsection 7.8, we will compare the area and delay between the decoders generated by our algorithm and the decoders written manually.

### 7.2. PCI Express 2.0 encoder

This PCI Express 2.0 [PCI-SIG 2009] encoder has 259 lines of verilog. Its input and output variables are shown in Table II. According to the 8b/10b encoding scheme [Widmer and Franaszek 1983], when $TXDATAK \equiv 0$, $TXDATA$ can be any value. But when $TXDATAK \equiv 1$, $TXDATA$ can only be 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD and FE. So we write an assertion to rule out those combinations not in this schema.

Algorithm 2 identifies $\vec{f} := CNTL\_TXEnable\_P0$ in 0.49 seconds. Algorithm 5 infers $valid(\vec{f}) := CNTL\_TXEnable\_P0$ in 1.21 seconds. Algorithm 6 obtains the minimized

Table II. The input and output variables of the PCI Express 2.0 encoder

| | variable name | width | description |
|---|---|---|---|
| Inputs | $TXDATA$ | 8 | The data to be encoded |
| | $TXDATAK$ | 1 | 1 means $TXDATA$ is a controlling character,<br>0 means $TXDATA$ is normal data |
| | $CNTL\_TXEnable\_P0$ | 1 | Indicating the validness of $TXDATA$ and $TXDATAK$ |
| Outputs | $HSS\_TXD$ | 10 | The encoded data |
| | $HSS\_TXELECIDLE$ | 1 | The electrical idle state |

$p := 4$, $l := 0$ and $r := 2$ in 0.68 seconds. Finally, generating the decoder's Boolean functions for $CNTL\_TXEnable\_P0$, $TXDATA$ and $TXDATAK$ costs 0.28 seconds. The decoder contains 156 gates and 0 registers with area 366 and delay 7.6.

The major breakthrough of this paper's algorithms is their ability to handle invalid data vector. So, it should be very interesting to show how the invalid data vector is mapped to output variable vector $\vec{o}$. By studying the source code of this encoder, we find that, when and only when $CNTL\_TXEnable\_P0 \equiv 0$ holds, that is, $TXDATA$ and $TXDATAK$ are invalid, the output electrical idle variable $HSS\_TXELECIDLE$ becomes 1. So, the decoder can use the output variable $HSS\_TXELECIDLE$ to uniquely determine the value of flow control variable $CNTL\_TXEnable\_P0$.

### 7.3. 10G Ethernet encoder XGXS

This encoder is compliant with clause 48 of IEEE 802.3 [IEEE 2012]. It has 214 lines of verilog. Its input and output variables are shown in Table III. This encoder also employs an 8b/10b encoding scheme [Widmer and Franaszek 1983] with two inputs: the 8-bit $encode\_data\_in$ to be encoded and 1-bit $konstant$ indicating a controlling character. According to the coding table in [Widmer and Franaszek 1983], when $konstant \equiv 0$, $encode\_data\_in$ can be of any value. But when $konstant \equiv 1$, $encode\_data\_in$ can only be 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD and FE. So, we write an assertion to exclude those combinations that aren't in this table.

Algorithm 2 identifies $\vec{f} := bad\_code$ in 0.31 seconds. Algorithm 5 infers $valid(\vec{f}) := bad\_code$ in 0.88 seconds. Algorithm 6 obtains the minimized $p := 4$, $l := 0$ and $r := 1$ in 0.52 seconds. Finally, generating the decoder's Boolean functions for $bad\_code$, $encode\_data\_in$ and $konstant$ costs 0.17 seconds. The decoder contains 163 gates and 0 registers with area 370 and delay 8.1.

Although this encoder uses the same coding mechanism as does the PCI Express 2.0 encoder mentioned above, the way it handles the invalid data vector is different. This encoder doesn't have a separate output variable to indicate the validness of the output data; instead, the validness and exact value of all input variables are both encoded in $encode\_data\_out$. By studying this encoder's source code, we find that when and only when $bad\_code \equiv 1$, that is, $encode\_data\_in$ and $konstant$ are invalid, the output variable $encode\_data\_out$ will become $0010111101$. So the decoder can use the output variable $encode\_data\_out$ to uniquely determine the flow control variable $bad\_code$.

Table III. The input and output variables of the 10G Ethernet encoder XGXS

| | variable name | width | description |
|---|---|---|---|
| Inputs | $encode\_data\_in$ | 8 | The data to be encoded |
| | $konstant$ | 1 | 1 means $encode\_data\_in$ is a special character,<br>0 means $encode\_data\_in$ is normal data |
| | $bad\_code$ | 1 | Indicating the validness of $konstant$ and $encode\_data\_in$ |
| Outputs | $encode\_data\_out$ | 10 | The encoded data |

Table IV. The input and output variables of the UltraSPARC T2 Ethernet encoder

|  | variable name | width | description |
|---|---|---|---|
| Inputs | $txd$ | 8 | The data to be encoded |
|  | $tx\_enc\_ctrl\_sel$ | 1 | Refer to Table V |
|  | $tx\_en$ | 1 | Transmission enable |
|  | $tx\_er$ | 1 | Transmitting an error character |
| Outputs | $tx\_10bdata$ | 10 | The encoded data |
|  | $txd\_eq\_crs\_ext$ | 10 | Transmitting an special error character with $tx\_er \equiv 1$ and $txd \equiv 8'h0F$ |
|  | $tx\_er\_d$ | 1 | Transmitting an error character |
|  | $tx\_en\_d$ | 1 | Transmission enable |
|  | $pos\_disp\_tx\_p$ | 1 | Indicating positive parity |

### 7.4. UltraSPARC T2 Ethernet encoder

This encoder comes from the UltraSPARC T2 open source processor. It is compliant with clause 36 of IEEE 802.3 standard [IEEE 2012], and has 864 lines of verilog.

Its input and output variables are shown in Table IV. This encoder also employs an 8b/10b encoding scheme [Widmer and Franaszek 1983], but with yet another flow control mechanism that is significantly different from that of the above two encoders. The data to be encoded is the 8-bit $txd$, but there is no standalone variable to indicate the control symbol. But only a 4-bit $tx\_enc\_ctrl\_sel$ used to define the action to be performed, as shown in Table V. Obviously, the functionalities of the control symbol indication and flow control mechanism are combined in $tx\_enc\_ctrl\_sel$. The last four cases in Table V can never be uniquely determined, because they cannot be distinguished from the case of ‘PCS_ENC_DATA. So we write an assertion to rule them out.

Algorithm 2 identifies $\vec{f} := \{tx\_enc\_ctrl\_sel, tx\_en, tx\_er\}$ in 3.76 seconds. Algorithm 5 infers $valid(\vec{f}) := tx\_enc\_ctrl\_sel \equiv$ ‘$PCS\_ENC\_DATA$ in 21.53 seconds. Algorithm 6 obtains the minimized $p := 5$,$l := 0$ and $r := 4$ in 6.15 seconds. Finally, generating the decoder's Boolean functions for $txd$, $tx\_enc\_ctrl\_sel$, $tx\_en$ and $tx\_er$ costs 3.40 seconds. The decoder contains 401 gates and 9 registers with area 920 and delay 10.2.

As shown in the last column of Table V, the first 5 cases have their own particular control symbol values assigned to $tx\_10bdata$, so the decoder can recover the value of the flow control variable $tx\_enc\_ctrl\_sel$ from $tx\_10bdata$.

### 7.5. Comparing our algorithm with the state-of-the-art algorithm on circuits without flow control mechanism

Table VI compares our algorithm to [Liu et al. 2012] on those benchmarks without flow control mechanism. For the 6 benchmarks from XFI to HM(15,11), we have their source code. And [Liu et al. 2012] have its experimental result. So we can compare the result of our algorithm to that of [Liu et al. 2012].

Table V. Actions to be performed in UltraSPARC T2 Ethernet encoder

| The name of action | The meaning of action |
|---|---|
| ‘PCS_ENC_K285 | sending K28.5 control symbol |
| ‘PCS_ENC_SOP | sending K27.7 control symbol |
| ‘PCS_ENC_T_CHAR | sending K29.7 control symbol |
| ‘PCS_ENC_R_CHAR | sending K23.7 control symbol |
| ‘PCS_ENC_H_CHAR | sending K30.7 control symbol |
| ‘PCS_ENC_DATA | sending the encoded txd |
| ‘PCS_ENC_IDLE2 | sending D16.2 data symbol following K28.5 |
| ‘PCS_ENC_IDLE1 | sending D5.6 data symbol |
| ‘PCS_ENC_LINK_CONFA | sending D21.5 data symbol following K28.5 |
| ‘PCS_ENC_LINK_CONFB | sending D2.2 data symbol following K28.5 |

Table VI. Comparing our algorithm with [Liu et al. 2012]

| Names | Our algorithm | | | | [Liu et al. 2012] | | |
|---|---|---|---|---|---|---|---|
| | time check. decoder. exist. | time gen. decoder | decoder area | decoder delay | time check. decoder exist. and gen. decoder | decoder area | decoder delay |
| XFI | 17.74 | 6.13 | 3878 | 13.8 | 8.59 | 3913 | 12.5 |
| SCRAMBLER | 2.55 | 0.55 | 698 | 3.8 | 0.42 | 640 | 3.8 |
| CC_3 | 0.06 | 0.03 | 116 | 8.5 | 0.21 | 104 | 9.1 |
| CC_4 | 0.16 | 0.09 | 490 | 13.3 | 0.20 | 129 | 9.0 |
| HM(7,4) | 0.09 | 0.03 | 258 | 8.1 | 0.05 | 255 | 7.3 |
| HM(15,11) | 1.49 | 2.23 | 5277 | 13.7 | 2.02 | 3279 | 13.2 |

By comparing the sum of the 2nd and 3rd columns to the 6th column, we can find that [Liu et al. 2012] is much faster than our algorithm, especially for the 2nd column. I think this is caused by the fact that our algorithm need to spend lots of time to check whether each input variables $i \in \vec{i}$ can be uniquely determined, while [Liu et al. 2012] can check all input variables with one SAT solving.

For most benchmarks, our decoder's area and delay are similar to [Liu et al. 2012]. For HM(15,11), our decoder is much larger than [Liu et al. 2012]. This is caused by the immature implementation of our interpolant algorithm, and can be significantly improved by porting similar codes from other formal tools. For CC_4, our decoder is also much larger and slower than [Liu et al. 2012], we will explain this issue in the following two subsections with more details.

### 7.6. Comparing run time overhead of increasing $p$, $l$ and $r$ concurrently and separately

Algorithm 3 increases $p$, $l$ and $r$ concurrently, and then reduce them with Algorithm 6. We call it A1 case. Subsection 5.2 shows another possibility that increases $p$, $l$ and $r$ separately. We call it A2 case. We compare these two cases in Table VII.

By comparing the total run time in column 6 and 11, obviously A2 is faster than A1 in most cases. But this doesn't mean that we should use A2 instead of A1.

According to Subsection 5.2, A1 needs to call SAT solver for $O(n)$ times, with $n = max(p, l, r)$. And A2 need to call SAT solver for $O(n^3)$ times. For benchmarks with small $n$, these two approaches don't have too much difference on this overhead. But for benchmarks with larger $n$, such as T2Eth, their difference is significant.

A2 beats A1 in smaller circuits, while A1 wins on larger ones. So we chose A1.

CC_4 is the only benchmark with different $p$, $l$ and $r$ on the 2nd and 7th column. This is caused by the different orders in expanding and reducing $l$ and $r$. For the A1 case, its decoder contains 14 registers, 206 gates, 490 area and 13.3 delay. For the A2 case, its decoder contains 10 registers, 61 gates, 154 area and 9.6 delay, which is comparable to that of [Liu et al. 2012] in Table VI. So A2 case is much better than A1. But this still

Table VII. Comparing run time overhead of increasing $p$, $l$ and $r$ concurrently and separately

| benchmarks | A1:Increasing concurrently | | | | | A2:Increasing separately | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | p,l,r | time find. $\vec{f}$ | time infer. $valid(\vec{f})$ | time min. $p$,$l$,$r$ | total run time | p,l,r | time find. $\vec{f}$ | time infer. $valid(\vec{f})$ | time min. $p$,$l$,$r$ | total run time |
| PCIE2 | 4,0,2 | 0.49 | 1.21 | 0.68 | 2.38 | 4,0,2 | 0.38 | 0.80 | 0.38 | 1.60 |
| XGXS | 4,0,1 | 0.31 | 0.88 | 0.52 | 1.71 | 4,0,1 | 0.23 | 0.58 | 0.30 | 1.11 |
| T2Eth | 5,0,4 | 3.76 | 21.53 | 6.15 | 31.44 | 5,0,4 | 15.47 | 13.85 | 6.19 | 35.51 |
| XFI | 3,1,0 | 4.59 | 3.60 | 9.55 | 17.74 | 3,1,0 | 3.52 | 2.75 | 10.05 | 16.32 |
| SCRAMBLER | 3,1,0 | 0.64 | 0.58 | 1.33 | 2.55 | 3,1,0 | 0.48 | 0.43 | 1.47 | 2.38 |
| CC_3 | 3,2,2 | 0.01 | 0.01 | 0.04 | 0.06 | 3,2,2 | 0.01 | 0.01 | 0.01 | 0.03 |
| CC_4 | 4,4,3 | 0.07 | 0.01 | 0.08 | 0.16 | 4,1,4 | 0.16 | 0.01 | 0.07 | 0.25 |
| HM(7,4) | 3,0,0 | 0.02 | 0.01 | 0.07 | 0.09 | 3,0,0 | 0.01 | 0.01 | 0.04 | 0.06 |
| HM(15,11) | 3,0,0 | 0.22 | 0.05 | 1.21 | 1.49 | 3,0,0 | 0.34 | 0.04 | 0.58 | 0.96 |

Table VIII. Comparing run times, areas and delay between the two approaches with and without minimizing $p$, $l$ and $r$ by Algorithm 6

| bench-marks | Not minimizing $l$ and $r$ with Alg. 6 | | | | | Minimizing $l$ and $r$ with Alg. 6 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | p,l,r | runtime generat. decoder | dec-oder area | # of reg | max logic delay | time min. $p$,$l$,$r$ | p,l,r | runtime generat. decoder | dec-oder area | # of reg | max logic delay |
| PCIE2 | 4,4,4 | 0.44 | 382 | 11 | 7.5 | 0.68 | 4,0,2 | 0.28 | 366 | 0 | 7.6 |
| XGXS | 4,4,4 | 0.35 | 351 | 20 | 8.2 | 0.52 | 4,0,1 | 0.17 | 370 | 0 | 8.1 |
| T2Eth | 5,5,5 | 4.76 | 1178 | 9 | 10.9 | 6.15 | 5,0,4 | 3.40 | 920 | 9 | 10.2 |
| XFI | 3,3,3 | 10.67 | 5079 | 190 | 16.50 | 9.55 | 3,1,0 | 6.13 | 3878 | 58 | 13.8 |
| SCRAMB. | 3,3,3 | 1.27 | 826 | 186 | 3.8 | 1.33 | 3,1,0 | 0.55 | 698 | 58 | 3.8 |
| CC_3 | 3,3,3 | 0.04 | 117 | 11 | 9.2 | 0.04 | 3,2,2 | 0.03 | 116 | 9 | 8.5 |
| CC_4 | 4,4,4 | 0.05 | 154 | 10 | 9.6 | 0.08 | 4,4,3 | 0.09 | 365 | 14 | 12.5 |
| HM(7,4) | 3,3,3 | 0.05 | 262 | 21 | 7.2 | 0.07 | 3,0,0 | 0.03 | 258 | 0 | 8.1 |
| HM(15,11) | 3,3,3 | 2.98 | 5611 | 45 | 13.5 | 1.21 | 3,0,0 | 2.23 | 5277 | 0 | 13.7 |

doesn't mean we need to increasing $p$, $l$ and $r$ separately. We will talk about this in the next subsection again and explain why.

### 7.7. Comparing run times, areas and delay between the two approaches with and without minimizing $l$ and $r$ by Algorithm 6

The experimental results are presented in Table VIII. When Algorithm 6 isn't used, the 2nd to 6th columns give respectively the $p$, $l$ and $r$ valuations, the runtime to generate the decoder, the decoder area, the number of registers, and the maximal logic delay. When Algorithm 6 is used, these experimental results are again presented in the last 5 columns. While the 7th column presents the time used to minimize $l$ and $r$.

By comparing the 2-6 columns with the 8-12 columns, it is obvious that the decoders area and number of registers are significantly reduced, with significant runtime overhead in reducing $l$ and $r$ shown in the 7th column.

Please notice the CC_4 benchmark again, from the 4th to 6th column, we can find that the area and delay is similar to A2 case in the previous subsection, but its value of $p$, $l$ and $r$ is similar to the A1 case. So this means that the decoder of CC_4 has at least three significantly different implementations with area 154 in A2 case in previous subsection, area 365 in the 10th column of Table VIII and area 490 in A1 case of previous subsection. The smaller one with area 154 and delay 9.6 is comparable to that of [Liu et al. 2012] shown in Table VI.

Which decoder implementation finally generated for CC_4 is determined by some instable implementation detail in the SAT solver and the Craig interpolant algorithm. So this answer the confusion in the previous subsection that, the circuit quality degradation in the A1 case isn't caused by increasing $p$, $l$ and $r$ concurrently. We should still increase them concurrently and then minimize them with Algorithm 6.

### 7.8. Comparing area and delay for our generated decoders and manually written decoders

The experimental results are presented in Table IX. All our decoders are faster and smaller (or at least similar to) than the manually ones.

Table IX. Comparing area and delay for the decoders generated by our algorithm and manually written decoders

| bench-marks | Decoder generated by us | | Manually written decoder | |
|---|---|---|---|---|
| | area | max logic delay | area | max logic delay |
| PCIE2 | 366 | 7.6 | 594 | 9.7 |
| XGXS | 370 | 8.1 | 593 | 11.0 |
| T2Eth | 920 | 10.2 | 764 | 11.7 |
| XFI | 3878 | 13.8 | 3324 | 28.1 |
| SCRAMBLER | 698 | 3.8 | 1035 | 6.4 |

## 8. RELATED PUBLICATIONS

### 8.1. Complementary synthesis

The first complementary synthesis algorithm was proposed by [Shen et al. 2009]. It checks the decoder's existence by iteratively increasing the bound of unrolled transition function sequence, and generates the decoder's Boolean function by enumerating all satisfying assignments of the decoder's output. Its major shortcomings are that it may not halt and that it has large runtime overhead in building the decoder.

The halting problem was independently tackled in [Shen et al. 2011] and [Liu et al. 2011] by searching for loops in the state sequence, while the runtime overhead problem was addressed in [Shen et al. 2012; Liu et al. 2011] by interpolant [McMillan 2003].

Shen et al. [2012] automatically inferred an assertion for configuration pins, which can lead to the decoder's existence. It can be seen as a special case of Algorithm 5, with the restriction that the inferred assertion must hold on all steps. Our Algorithm 5, on the other hand, is the first algorithm that allows states with and without the inferred assertion to be interleaved freely with each other.

Tu and Jiang [2013] takes the encoder's initial states into consideration with property directed reachability analysis [Eén et al. 2011], so that the encoder's infinite history can be used to generate the decoder's output. This algorithm can handle some encoders that cannot be handled by the state-of-the-art algorithms.

### 8.2. Program inversion

Program inversion derives a program $P^{-1}$ that negates the computation of a given program $P$. So it is very similar to complementary synthesis.

The initial work on program inversion [Dijkstra 1979] used proof-based approaches, which could handle only very small programs and very simple syntax structures.

Glück and Kawabe. [2005] inverted first-order functional programs by eliminating nondeterminism with LR-based parsing methods. But the use of functional languages in that work is incompatible with our complementary synthesis.

Srivastava et al. [2011] assumed that an inverse program was related to the original program, so the space of possible inversions can be inferred by automatically mining the original program for expressions, predicates, and control flow. This algorithm inductively rules out invalid paths that can't fulfill the requirement of inversion until only the valid ones remain. So, it can't guarantee the correctness of its solution if its assumptions don't hold.

### 8.3. Satisfying Assignments Enumeration

Some algorithms try to enumerate all satisfying assignments faster by enlarging each complete satisfying assignment. McMillan [2002] constructs an alternative implication graph in SAT solver, which records the reasoning relation that leads to the assignment of a particular variable. All variables outside this graph can be ruled out from the complete assignment. In [Ravi and Somenzi 2004] and [Chauhan et al. 2004], those variables whose absence can't make $obj \equiv 0$ satisfiable are removed one by one. In [Shen et al. 2005] and [Jin and Somenzi: 2005], conflict analysis based approaches are used to remove multiple irrelevant variables in one SAT run. In [Grumberg et al. 2004], the variable set is divided into an important subset and an unimportant subset. Variables in the important subset have higher decision priority than those unimportant ones. Thus, the important subset forms a search tree, with each leaf being another search tree for the unimportant set. Ganai et al. [2004] qualifies out unimportant variables by setting them to constant value returned by the SAT solver.

Other algorithms constructs interpolations to cover more satisfying assignments. Jiang et al. [2009] constructs a first formula that contradicts with another formula, from which an interpolation can be derived and used as an over-approximation of the first formula. Chockler et al. [2012] generates interpolation with a framework similar to the iterative enumerating and enlarging approaches mentioned above. But there are two enlarging steps for each enumerated assignment, in which the assignments are enlarged with respect to the two formulas involved in constructing interpolant. It is the first paper that constructs interpolant without proof.

## 9. CONCLUSIONS

In this paper, we propose, for the first time, a framework to handle flow control mechanism in complementary synthesis problem. Experimental results indicate that our framework can always successfully handle many complex encoders from real industrial projects, such as PCI Express [PCI-SIG 2009] and Ethernet [IEEE 2012].

## ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

## ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their fruitful suggestions.

## REFERENCES

Dennis Abts and John Kim. 2011. *High Performance Datacenter Networks* (1st. ed.). Synthesis Lectures on Computer Architecture, Vol. 14. Morgan and Claypool, Chapter 1.6, 7–9. DOI:http://dx.doi.org/10.2200/S00341ED1V01Y201103CAC014

Berkeley Logic Synthesis and Verification Group 2008. ABC:A system for sequential synthesis and verification. (2008). http://www.eecs.berkeley.edu/alanmi/abc/.

Pankaj Chauhan, Edmund M. Clarke, and Daniel Kroening. 2004. A sat-based algorithm for reparameterization in symbolic simulation. In *Proceedings of the 41th Design Automation Conference, DAC 2004 (DAC '04)*. IEEE, 524–529.

Hana Chockler, Alexander Ivrii, and Arie Matsliah. 2012. Computing Interpolants without Proofs. In *8th International Haifa Verification Conference, HVC 2012*, Tanja E. J. Vos Armin Biere, Amir Nahir (Ed.). Lecture Notes in Computer Science, Vol. 7857. Springer-Verlag, Berlin Heidelberg, 72–85. DOI:http://dx.doi.org/10.1007/978-3-642-39611-3_12

William Craig. 1957. Linear reasoning: A new form of the herbrand-gentzen theorem. *The Journal of Symbolic Logic* 22, 3 (Sept. 1957), 250–268.

Edsger W. Dijkstra. 1979. Program Inversion. In *Proceeding of Program Construction, International Summer School*. Springer-Verlag, London, UK, 54–57.

Niklas Eén, Alan Mishchenko, and Robert K. Brayton. 2011. Efficient implementation of property-directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design,FMCAD 2011 (FMCAD '11)*. FMCAD Inc, Austin, TX, USA, 125–134.

Niklas Eén and Niklas Sörensson. 2003. An extensible sat-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003*, Armando Tacchella Enrico Giunchiglia (Ed.). Lecture Notes in Computer Science, Vol. 2919. Springer-Verlag, Berlin Heidelberg, 502–518. DOI:http://dx.doi.org/10.1007/978-3-540-24605-3_37

Malay K. Ganai, Aarti Gupta, and Pranav Ashar. 2004. Efficient sat-based unbounded symbolic model checking using circuit cofactoring. In *Proceedings of the 2004 International Conference on Computer-Aided Design, ICCAD 2004 (ICCAD '04)*. ACM, 510–517. DOI:http://dx.doi.org/10.1109/ICCAD.2004.1382631

Robert Glück and Masahiko Kawabe. 2005. A method for automatic program inversion based on LR(0) parsing. *Journal Fundamenta Informaticae* 66, 4 (January 2005), 367–395. DOI:http://dx.doi.org/10.1109/TCAD.2012.2191288

Orna Grumberg, Assaf Schuster, and Avi Yadgar. 2004. Memory efficient all-solutions sat solver and its application for reachability analysis. In *International Conference on Formal Methods in Computer-Aided Design,FMCAD 2011*, Andrew K. Martin Alan J. Hu (Ed.). Lecture Notes in Computer Science, Vol. 3312. Springer-Verlag, Berlin Heidelberg, 275–289. DOI:http://dx.doi.org/10.1007/978-3-540-30494-4_20

IEEE. 2012. IEEE Standard for Ethernet SECTION FOURTH. (2012). Retrieved January 25, 2013 from http://standards.ieee.org/getieee802/download/802.3-2012_section4.pdf

Jie-Hong Roland Jiang, Hsuan-Po Lin, and Wei-Lun Hung. 2009. Interpolating functions from large Boolean relations. In *Proceedings of 2009 International Conference on Computer-Aided Design (ICCAD '09)*. IEEE, 779–784.

HoonSang Jin and Fabio Somenzi:. 2005. Prime clauses for fast enumeration of satisfying assignments to boolean circuits. In *Proceedings of the 42th Design Automation Conference, DAC 2005 (DAC '05)*. IEEE, 750–753. DOI:http://dx.doi.org/10.1109/DAC.2005.193911

Hsiou-Yuan Liu, Yen-Cheng Chou, Chen-Hsuan Lin, and Jie-Hong R. Jiang. 2011. Towards completely automatic decoder synthesis. In *Proceedings of the 2011 International Conference on Computer-Aided Design, ICCAD 2011 (ICCAD '11)*. IEEE Press, San Jose, CA, USA, 389–395. DOI:http://dx.doi.org/10.1109/ICCAD.2011.6105359

Hsiou-Yuan Liu, Yen-Cheng Chou, Chen-Hsuan Lin, and Jie-Hong R. Jiang. 2012. Automatic Decoder Synthesis: Methods and Case Studies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 9 (September 2012), 31:1319–31:1331. DOI:http://dx.doi.org/10.1109/TCAD.2012.2191288

Kenneth L. McMillan. 2002. Applying sat methods in unbounded symbolic model checking. In *International Conference on Computer Aided Verification, CAV 2002*, Kim Guldstrand Larsen Ed Brinksma (Ed.). Lecture Notes in Computer Science, Vol. 2404. Springer-Verlag, Berlin Heidelberg, 250–264. DOI:http://dx.doi.org/10.1007/3-540-45657-0_19

Kenneth L. McMillan. 2003. Interpolation and sat-based model checking. In *Computer Aided Verification, 15th International Conference, CAV 2003*, Fabio Somenzi Warren A. Hunt Jr. (Ed.). Lecture Notes in Computer Science, Vol. 2725. Springer-Verlag, Berlin Heidelberg, 1–13. DOI:http://dx.doi.org/10.1007/978-3-540-45069-6_1

PCI-SIG. 2009. PCI Express Base 2.1 Specification. (2009). Retrieved January 25, 2013 from http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r2_1_04Mar09.pdf

Kavita Ravi and Fabio Somenzi. 2004. Minimal assignments for bounded model checking. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004*, Andreas Podelski Kurt Jensen (Ed.). Lecture Notes in Computer Science, Vol. 2988. Springer-Verlag, Berlin Heidelberg, 31–45. DOI:http://dx.doi.org/10.1007/978-3-540-24730-2_3

ShengYu Shen, Ying Qin, and Sikun Li. 2005. Minimizing counterexample with unit core extraction and incremental sat. In *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005*, Radhia Cousot (Ed.). Lecture Notes in Computer Science, Vol. 3385. Springer-Verlag, Berlin Heidelberg, 298–312. DOI:http://dx.doi.org/10.1007/978-3-540-30579-8_20

ShengYu Shen, Ying Qin, KeFei Wang, Zhengbin Pang, Jianmin Zhang, and Sikun Li. 2012. Inferring Assertion for Complementary Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 8 (August 2012), 31:1288–31:1292. DOI:http://dx.doi.org/10.1109/TCAD.2012.2190735

ShengYu Shen, Ying Qin, KeFei Wang, LiQuan Xiao, Jianmin Zhang, and Sikun Li. 2010. Synthesizing Complementary Circuits Automatically. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 8 (August 2010), 29:1191–29:1202. DOI:http://dx.doi.org/10.1109/TCAD.2010.2049152

ShengYu Shen, Ying Qin, LiQuan Xiao, KeFei Wang, Jianmin Zhang, and Sikun Li. 2011. A Halting Algorithm to Determine the Existence of the Decoder. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 10 (October 2011), 30:1556–30:1563. DOI:http://dx.doi.org/10.1109/TCAD.2011.2159792

ShengYu Shen, Jianmin Zhang, Ying Qin, and Sikun Li. 2009. Synthesizing complementary circuits automatically. In *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*. IEEE Press, San Jose, CA, USA, 381–388. DOI:http://dx.doi.org/10.1145/1687399.1687472

Saurabh Srivastava, Sumit Gulwani, Swarat Chaudhuri, and Jeffrey S. Foster. 2011. Path-based inductive synthesis for program inversion. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation,PLDI 2011 (PLDI '11)*. ACM Press, San Jose, CA, USA, 492–503. DOI:http://dx.doi.org/10.1145/1993498.1993557

Kuan-Hua Tu and Jie-Hong R. Jiang. 2013. Synthesis of feedback decoders for initialized encoders. In *Proceedings of the 50th Annual Design Automation Conference, DAC 2013 (DAC '13)*. ACM Press, Austin, TX, USA, 1–6. DOI:http://dx.doi.org/10.1145/2463209.2488794

Al X. Widmer and Peter A. Franaszek. 1983. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code. *IBM Journal of Research and Development* 27, 5 (May 1983), 440–451. DOI:http://dx.doi.org/10.1147/rd.275.0440

Received February 2014; revised March 2014; accepted June 2014

First thank you for your hard work in reviewing this paper. I find the review comments are very helpful, and I have taken these comments into consideration, and polish my paper significantly.

According to the TODAES new pages limit, I shrink my paper from 30 pages to 25pages in the following way:

1 in Section 8 “RELATED PUBLICATIONS”, I remove the two subsections “Protocol converter synthesis” and “Logic synthesis with Craig interpolation” as their relevance are somewhat relative low.

2 I also rephrase many paragraphs without removing them, such that they occupy less space.

I present below the reviewer comments in BLUE color, and my response in RED color.

---------------------- REVIEWER COMMENTS --------------

Referee: 1

Comments to the Author

The revision addressed most of my comments. Thank you. However, there are several minor clarifications needed.

1. Regarding the over-approximation issue of interpolants, you commented “in the special case that A and B are complement to each other, that is, every assignment must satisfy either A or B but not both, then this interpolation must cover and only cover A.” Please explain why A and B have to be complementary to each other in your formulas. Essentially why there is no assignment a* under which R(a*, b, 0) and R(a*, b, 1) are both unsatisfiable? That is, why should R(a, b, t) be satisfiable under any assignment on variable a?

You are right, I have not described it clearly here.

Actually this is an assumption for this algorithm. It will NOT be fulfilled by the algorithm itself, but instead should be fulfilled by the actual application of the algorithm.

So I add three paragraph in page 9, subsection 4.1, the 2nd to 4th paragraphs:

++++++++++++++++++++++++++++++++++++++++

(1) Assumption 1 : R(a, b, 0) ∧ R(a, b, 1) is unsatisfiable. a and b are respectively called the important and the non-important variable vectors, t is the target variable.

(2) Assumption 2 : R(a, b, t) is satisfiable for all valuations of a.

In the remainder of this paper, when we use the algorithm introduced in this subsection, we will show that these two assumptions are fulfilled.

++++++++++++++++++++++++++++++++++++++++

In page 11, paragraph 3 starting with “That is” , this is the first application of the algorithm, we described how the two assumptions are fulfilled.

In page 12, paragraph 3 starting with “Obviously”, this is the second application of the algorithm, we describe how the two assumptions are fulfilled.

2. In (9), “d_{p+l} \neq d’_{p+l}”. You use “\neq” instead of “\not \equiv”. Please clarify whether “\neq” means nonequivalence in some bit or bitwise nonequivalence.

You are right, I have not described it clearly here.
On page 10, subsubsection 4.2.1, paragraph 2 starting with “Here,”, we have clarified that \neq means nonequivalence in some bit.

3. In page 2, “d can not be uniquely by…” -> “d can not be uniquely determined by…”; “d is not need by…” -> “d is not needed by…”; “can be any value” -> “can be of any value”

Thank you, I have changed it according to your suggestion.

4. In page 4, “it use…” -> “it uses…”

Thank you, I have changed it according to your suggestion. I also change some other similar typos.

5. In page 5, “p,l” -> “p, l”. There are many other places where space separation is missing.
Thank you, I have changed it according to your suggestion. I also change some other similar typos.

6. In page 5, “They can always be satisfiable with…” -> “They can always be satisfied with …”
Thank you, I have changed it according to your suggestion.

7. The first sentence of page 6 can be merged with the last paragraph of page 5.
Thank you, I have changed it according to your suggestion.

8. In page 6, “Second and more important,” -> “Second, and more importantly,”
Thank you, I have changed it according to your suggestion.

9. It is not clear which paper in references is referred to by the citation “[Jie-Hong Roland Jiang 2009]”.
Thank you, there is a typo in the bib file, I have corrected it.

10. The relevance of [Wu et al. 2010] to the paper is not well described. Please either describe ECO in detail or remove it if the relevance is low.

Thank you, you are right. I have remove it.

Referee: 2

Comments to the Author
thanks for the effort in revising.