Complementary Synthesis for Encoder with Flow Control Mechanism

YING QIN and SHENGYU SHEN and QINGBO WU and 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 encoder employ flow control mechanism that fail this assumption. Such encoders, when its output symbol sequence is too fast to be processed by the decoder, will stop transmitting data symbols, but instead transmitting an idle symbol that can only uniquely determine a subset of the encoder's input variables. And the decoder should recognize and discard this idle symbol. Although this mechanism can prevent losing data symbols, it fail the assumption of all complementary synthesis algorithms, because some input variables can not be uniquely determined by the idle symbol.

This paper proposes the first algorithm to handle such encoders. **First**, it identifies all input variables that can be uniquely determined, and take 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**, the decoder's Boolean function for flow control variables can be characterized with Craig interpolant. For other input variables, their decoder's Boolean function can also be characterized similarly but the inferred predicate must be enforced first.

Experimental results on several complex encoders indicate that our 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, Logic synthesis, Verification

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

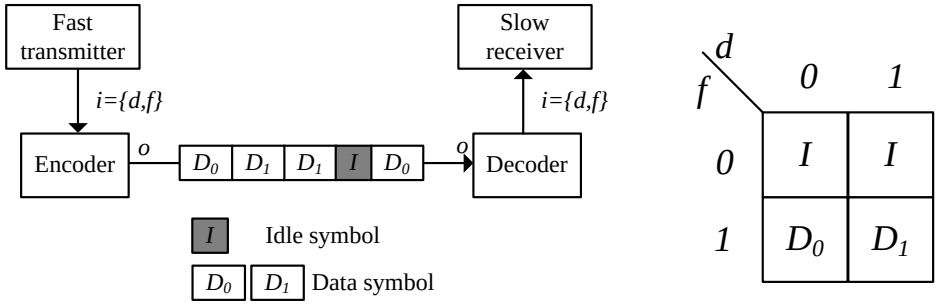
ACM Reference Format:
Ying Qin and ShengYu Shen and QingBo Wu and HuaDong Dai and Yan Jia, 2014. Complementary Synthesis for Encoder with Flow Control Mechanism. *ACM Trans. Des. Autom. Electron. Syst.* 9, 4, Article 39 (March 2010), 30 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

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] try to ease this job by automatically generating a decoder from**

This work was funded by projects 61070132 and 61133007 supported by National Natural Science Foundation of China, the 863 Project of China under contract 2012AA01A301.
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



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

an encoder's specification, with the assumption that $\vec{i}$ can always be uniquely determined by a bounded sequence of $\vec{o}$.

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 include 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 prevent 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 not 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.

This mechanism can prevent the faster transmitter from transmitting too much data that can not be handled by the slower receiver. But it fails the assumption of all current complementary synthesis algorithms [Shen et al. 2009;2010; 2011; 2012;Liu et al. 2011;2012;Tu and Jiang 2013], because d can not be uniquely by the idle symbol I. It is obvious that, to resolve this problem and generate the decoder, 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 is not need by the receiver and can be any value.

Thus, according to this insight, we propose in this paper the first complementary synthesis algorithm to handle encoders with flow control mechanism in three steps: **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 not be uniquely determined is called the data variables $\vec{d}$. **Second**, it infers a suffi-

cient and necessary predicate $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined by a bounded sequence of the encoder's output variables $\vec{o}$. **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 the 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 from real projects (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 presents the algorithm that 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 6 presents the algorithm to characterize 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 $\vec{v}$, that is $\vec{v} = (\ldots, v, \ldots)$, 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 $A : V \to B$, so that F can be evaluated to 1. If such a satisfying assignment 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 ϕ_A and ϕ_B, with $\phi_A \wedge \phi_B$ unsatisfiable, there exists a formula ϕ_I referring only to the common variables of ϕ_A and ϕ_B such that $\phi_A \Rightarrow \phi_I$ and $\phi_I \wedge \phi_B$ is unsatisfiable. We call ϕ_I the **interpolant**[Craig 1957] of ϕ_A with respect to ϕ_B and use McMillan's algorithm [McMillan 2003] to generate it.

2.2. Incremental SAT mechanism of MiniSat solver

In this paper, we use the MiniSat solver [Eén and Sörensson 2003] to solve the generated CNF formulas. Like many other SAT solver based on conflict driven learning [Zhang et al. 2001], MiniSat generates learned clauses from conflicts in searching, and records them to prevent the same conflict from rising again. This mechanism can significantly speedup a particular SAT solving.

In many applications, there often exists a serial of CNF formulas tightly related to each other. If the learned clauses can be shared between them, then these formulas can be solved much faster.

MiniSat provides an incremental SAT mechanism to reuse these learned clauses. This mechanism includes two procedures:

(1) $addClause(F)$ used to add a CNF formula F to the clause database of MiniSat.
(2) $solve(A)$ that takes a set of literals A as assumptions, and solves the CNF formula $F \wedge \bigwedge_{a \in A} a$.

2.3. 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 $\vec{s}_n \equiv \vec{s}_m$.

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

All the state-of-the-art complementary synthesis algorithms [Shen et al. 2009;2010; 2011; 2012;Liu et al. 2011;2012;Tu and Jiang 2013] assume that $\vec{i}$ can be uniquely determined, so they always take $\vec{i}$ as a whole, and never consider individual variables $i \in \vec{i}$. But in this paper, we need to check each $i \in \vec{i}$ one by one, so there may be minor differences between our presentation here and that of [Shen et al. 2009;2010; 2011; 2012;Liu et al. 2011;2012;Tu and Jiang 2013].

The first such halting algorithm is proposed in [Shen et al. 2011]. Its basic idea is to unroll the transition function into longer and longer length. And for each length, it use two approximated approaches to determine the answer, the first one is an underestimate one that presented in 2.4.1, while the second one is an overestimate one presented in 2.4.2. That is, when the first one says **YES** then the final answer is **YES**, and when the second approach says **NO** then the final answer is **NO**. And we will show in 2.4.3 that these two approaches will eventually converge and give conclusive answer at some particular unrolling length.

*2.4.1. **The underestimate 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 $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ that are on the left-hand and right-hand sides of i_{p+l}, which is 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 satisfiable 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 Figure 2, it is obvious that, if $F_{PC}(p,l,r)$ is unsatisfiable, then $F_{PC}(p',l',r')$ is also unsatisfiable with $p' \geq p$, $l' \geq l$ and $r' \geq r$. By studying Equation (1), we can find that the clause set of $F_{PC}(p',l',r')$ is a super set of $F_{PC}(p,l,r)$. This also lead to the same conclusion.

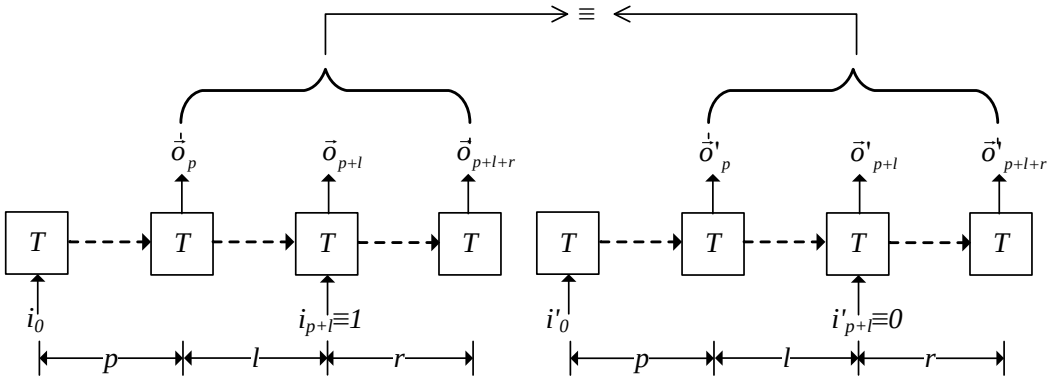


Fig. 2. The underestimate approach checking if i_{p+l} can be uniquely determined

This means, the bounded proof of $F_{PC}(p,l,r)$'s unsatisfiability can be generalized to all cases for larger p, l and r.

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) does not include an initial state, instead it use the p steps prefix state transition sequence $< \vec{s_0}, \ldots, \vec{s_{p-1}} >$ to propagate the constraints $assertion(\vec{i})$ into the state sequence $< \vec{s_p}, \ldots, \vec{s_{p+l+r}} >$, such that some states that can not 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. A breakthrough algorithm was proposed in [Tu and Jiang 2013] to rule out unreachable states by inferring inductive invariants. But this algorithm can not handle our most complex XFI benchmark [Shen et al. 2011], while our algorithms always can. Second and more important, ignoring initial states improve the decoder's reliability by making the decoder's output depend on only bounded history of its input. Thus any corrupted $\vec{o}$ fed to the decoder 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 not happen on all our benchmarks from industrial applications.

2.4.2. *The overestimate approach*. .

In the last subsection, we have just learned that, if $F_{PC}(p,l,r)$ is unsatisfiable, then i_{p+l} can be uniquely determined for all larger p, l and r. **On the other hand**, if $F_{PC}(p,l,r)$ is satisfiable, then i_{p+l} cannot be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for this particular valuation of p, l and r. There are two possible cases:

(1) i_{p+l} can be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ for larger p, l and r;
(2) i_{p+l} can not 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 first case, then by iteratively increasing p, l and r, $F_{PC}(p,l,r)$ will eventually become unsatisfiable. But if it is the second case, then this iterative algorithm will never terminate.

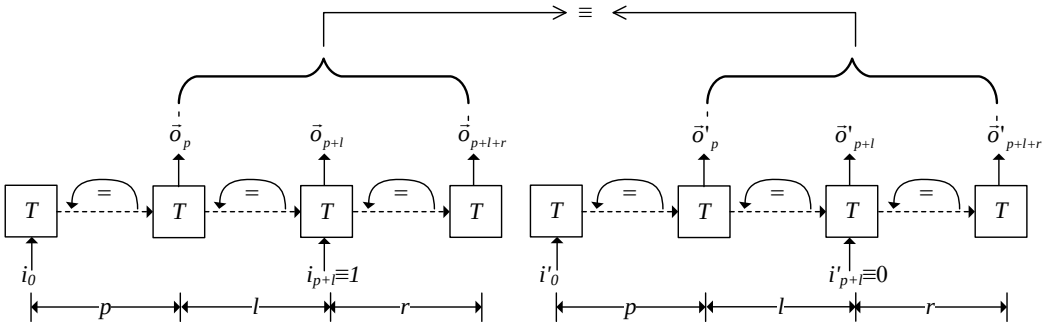


Fig. 3. The overestimate approach checking if i_{p+l} can NOT be uniquely determined

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 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.

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

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 variables can NOT be uniquely determined.

When $F_{LN}(p,l,r)$ is satisfiable, then i_{p+l} can **NOT** be uniquely determined by $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. **More importantly, by unrolling these three loops, we can also make $F_{LN}(p,l,r)$ satisfiable for 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 any larger p, l and r.*

2.4.3. ***The full algorithm***. .

With Propositions 2.1 and 2.2, we can generalize their bounded proof to unbounded cases. Thus, we can build the halting Algorithm 1 that determines if there exists p, l and r that enable an input variable i_{p+l} to be uniquely determined by the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$:

(1) On the one hand, if there actually exists such p, l and r, then eventually $F_{PC}(p,l,r)$ will become unsatisfiable in Line 4;
(2) On the other hand, if there does not 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.

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

Input: The input variable vector $\vec{i}$.
Output: $\vec{f} \subset \vec{i}$ and $\vec{d} \subset \vec{i}$ that can and can not be uniquely determined ,and the maximal p, l and r .

```
1  f⃗ := {}; d⃗ := {};
2  p := 0 ; l := 0 ; r := 0 ;
3  while i⃗ ≠ {} do
4      p++;  l++;  r++;
5      assume i ∈ i⃗;
6      if F_PC(p, l, r) is unsatisfiable for i then
7          f⃗ := i ∪ f⃗;
8          i⃗ := i⃗ − i;
9      else if F_LN(p, l, r) is satisfiable for i and assume its satisfying assignment is A then
10         foreach j ∈ i⃗ do
11             if A(j_{p+l}) ≠ A(j'_{p+l}) then
12                 i⃗ := i⃗ − j;
13                 d⃗ := j ∪ d⃗
14
15 return (f⃗, p_max, l_max, r_max)
```

3. IDENTIFYING FLOW CONTROL VARIABLES

We will first introduce how to find out the set of flow control variables in Subsection 3.1, and then introduce how to speed up this algorithm with incremental SAT in Subsection 3.2.

3.1. Finding out flow control variables

To facilitate the presentation of our algorithm, we assume that the input variable vector $\vec{i}$ can be partitioned into two vectors: the flow control vector $\vec{f}$ and the data vector $\vec{d}$. And the algorithm to find out this partition will be presented in this Subsection.

The flow control vector $\vec{f}$ are used to represent 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 variable 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 1, the initial value of f and d are set to empty vector. At Line 2, the initial value of p, l and r are all set to 0.

At Line 3, a while loop is used to iterate on all $i \in \vec{i}$.

At Line 7, the input variable i that can be uniquely determined will be added to the 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 9, 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 11, 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 do not harm our overall framework, because the decoder's Boolean function 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 2.2.

$F_{PC}(p,l,r)$ in Equation 1 can be partitioned into the following two equations:

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

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

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

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

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 not be uniquely determined, and the maximal p, l and r .

```
1  f⃗ := {}; d⃗ := {};
2  p:= 0 ; l:= 0 ; r:= 0 ;
3  while i⃗ ≠ {} do
4      p++; l++; r++;
5      addClause(C_PC(p,l,r));
6      foreach i ∈ i⃗ do
7          if solve(A_PC(p,l,r)) is unsatisfiable for i then
8              i⃗ := i⃗ − i;
9              f⃗ := i ∪ f⃗;
10     addClause(C_LN(p,l,r));
11     foreach i ∈ i⃗ do
12         if solve(A_LN(p,l,r)) is satisfiable for i and satisfying assignment is A then
13             foreach j ∈ i⃗ do
14                 if A(j_{p+l}) ≠ A(j'_{p+l}) then
15                     i⃗ := i⃗ − j;
16                     d⃗ := j ∪ d⃗
17 return (f⃗,p,l,r)
```

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

It is obvious that 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 $addClause(C_{LN})$. 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 5 and 10, and the two new $solve$ in Line 7 and 12. They are the procedures provided by MiniSat's incremental SAT mechanism mentioned in Subsection 2.2.

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 enable $\vec{d}$ to be uniquely determined by a bounded sequence of $\vec{o}$.

4.1. Characterizing a function that makes a Boolean formula satisfiable

Assume that $R(\vec{a},\vec{b},t)$ is a Boolean formula with $R(\vec{a},\vec{b},0) \wedge R(\vec{a},\vec{b},1)$ unsatisfiable. $\vec{a}$ and $\vec{b}$ are respectively called the important and the non-important variable vectors, while t is the target variable.

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 Craig interpolant $ITP(\vec{a})$ of $R(\vec{a},A(\vec{b}),1)$ with respect to $R(\vec{a},A(\vec{b}),0)$ can be computed and 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

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 ϕ_A with respect to ϕ_B ;
7 $FSAT_R(\vec{a}) := ITP(\vec{a}) \vee FSAT_R(\vec{a})$;
8 **return** $FSAT_R(\vec{a})$

make $R(\vec{a}, \vec{b}, 1)$ satisfiable, but has not 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, to remove $\vec{b}$ from them.

Thus, $\phi_A \wedge \phi_B$ in Line 6 is unsatisfiable, and the common variables vector of ϕ_A and ϕ_B is $\vec{a}$. So a Craig interpolant $ITP(\vec{a})$ can be generated with the McMillian'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

In this Section, we will present how to compute the predicate $valid(\vec{f})$ that enables $\vec{d}$ to be uniquely determined. Similar to the halting algorithm presented in Subsection 2.4, we will first present how to compute a monotonically growing under-approximation of $valid(\vec{f})$ in 4.2.1, and them present how to compute a monotonically shrinking over-approximation of $valid(\vec{f})$ in 4.2.2, and finally show that they will converge to $valid(\vec{f})$ in 4.2.3.

4.2.1. *Computing monotonically growing under-approximation of* $valid(\vec{f})$. .

By replacing i in Equation (1) with $\vec{d}$, we have:

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

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 a new formula $T_{PC}(p,l,r)$ by collecting the 3rd line of Equation (9):

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

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

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

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 $F'^d_{PC}(p,l,r,t)$. 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 grow 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:

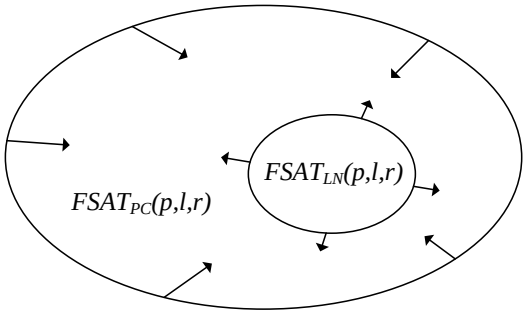


Fig. 4. The monotonicity of $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$

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

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):

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

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

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

Obviously $F^d_{LN}(p,l,r)$ and $F'^d_{LN}(p,l,r,1)$ are equivalent. 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

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

```
1 p:= p_max; l:= l_max; r:= r_max ;
2 while ¬FSAT_LN(p,l,r) ∧ FSAT_PC(p,l,r) is satisfiable do
3     p ++ ; l ++ ; r ++ ;
4 end
5 return ¬FSAT_LN(p,l,r)
```

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

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 $FSAT_{LN}(p,l,r)$ 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 $FSAT_{LN}(p,l,r)$ are equal to each other. 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 Boolean 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, we know that 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 we know that $A(\vec{f}_{p'+l'}) \equiv A'(\vec{f}_{p'+l'})$.

Intuitively, as shown in Figure 5, we can map the valuations of 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 $p'+l'-l-p \leq n \leq p'+l'+r$, 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 $A''' \equiv A$.

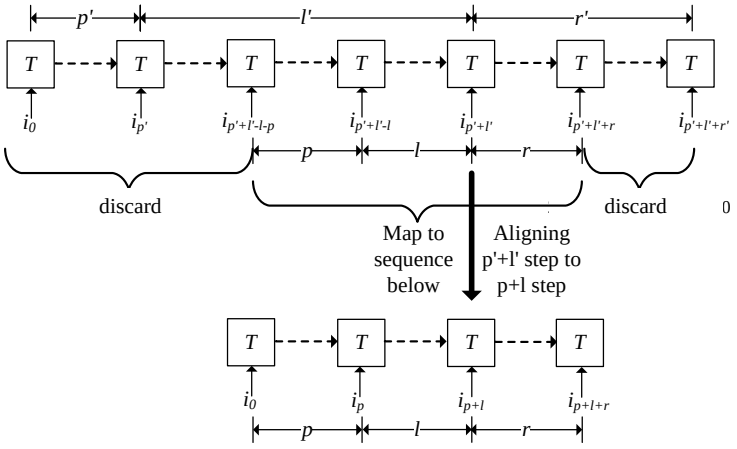


Fig. 5. Mapping $F'^{d}_{PC}(p', l', r', 1)$ to $F'^{d}_{PC}(p, l, r, 1)$ by aligning $(p' + l')$-step to $(p + l)$-step.

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 $vecf_{p'+l'}$, we can get the fourth satisfying assignment $A''' : \vec{f}_{p'+l'} \rightarrow B$. It is obvious that $A \equiv A'''$.

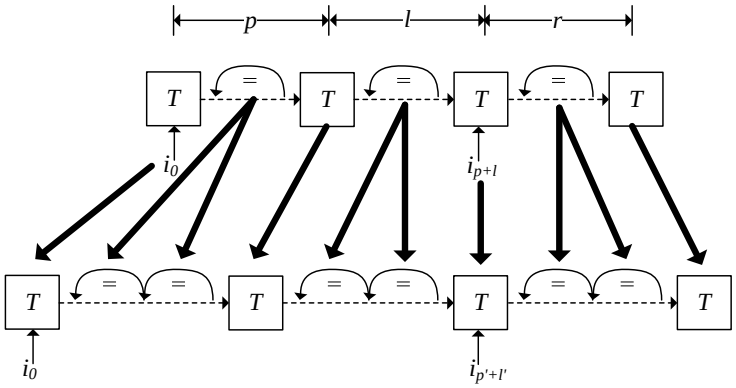


Fig. 6. Mapping $F'^{d}_{LN}(p, l, r, 1)$ to $F'^{d}_{LN}(p', l', r', 1)$ by aligning $(p + l)$-step to $(p' + l')$-step.

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 does not 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. Let's first prove the covering case. $FSAT_{LN}(p,l,r)$ covers a set of valuations 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.

We then prove the only covering case. 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. 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) $FSAT_{LN}(p,l,r)$ does not cover A. It is the last $FSAT_{LN}(p,l,r)$ computed by Algorithm 5. 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 does not cover A.

These two cases conflit 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 VALUATION OF L AND R

We will 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 the one in Subsection 5.2.

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

```
for r' := r → 1 do
    if F_PC(p, l, r' − 1) ∧ valid(f⃗_{p+l}) is satisfiable then
        break
    else if r' ≡ 1 then
        r' := r' − 1
        break
for l' := l → 1 do
    if F_PC(p, l' − 1, r') ∧ valid(f⃗_{p+l'−1}) is satisfiable then
        break
    else if l' ≡ 1 then
        l' := l' − 1
        break
return < l', r' >
```

5.1. Minimizing the valuation of l and r

As we have increase the value of p,l and r simultaneously in Algorithm 5, there may be some redundancy in the valuation of p,l and r, which may lead to unnecessary overhead in the decoder's circuit area and delay.

For example, assume the encoder is a simple buffer whose $o := i$. With $p \equiv 0$,$l \equiv 0$ and $r \equiv 0$, we can get the simplest decoder $i := o$. In such a decoder, there is only one buffer and no registers. But with $p \equiv 0$,$l \equiv 0$ and $r \equiv 1$, we need an additional register to delay o for one step, and then recover i_i from the delayed o.

According to Figure 2, it is obvious that, r affect both the decoder's latency and circuit area, l affect only the decoder's circuit area, while p does not affect the decoder.

So as shown in Algorithm 6 we chose to first minimize r, then minimize l.

5.2. An alternative approach for comparison

In above discussion, we first find out the flow control variables by increasing p, l and r simultaneously 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 simultaneously. 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 simultaneously and then minimized them with Algorithm 6 is much faster than increasing them separately. We will also explain the reason there.

6. CHARACTERIZING THE DECODER'S BOOLEAN FUNCTION

In Section 3, the encoder's input vector $\vec{i}$ has been partitioned into two vectors: the flow control vector $\vec{f}$ and the data vector $\vec{d}$. The algorithms to characterize the decoder's Boolean functions that compute $\vec{f}$ and $\vec{d}$ 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 ϕ_A with respect to ϕ_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 ϕ_A and ϕ_B is $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$. So, a Craig interpolant ITP can be derived by McMillian'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 ϕ_A and $f'_{p+l} \equiv 0$ from ϕ_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 $d \in \vec{d}$:

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

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

Each variable $d \in \vec{d}$ can be uniquely determined by the encoder's output only when $valid(\vec{f})$ holds. So, if $valid(\vec{f}_{p+l})$ holds, for each particular valuation of the encoder's output sequence $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$, d_{p+l} cannot be 0 and 1 at the same time. So, $\phi'_A \wedge \phi'_B$ is unsatisfiable. Thus, a Craig interpolant ITP can be derived by McMillian's algorithm[McMillan 2003] from the unsatisfiability proof of $\phi'_A \wedge \phi'_B$, which covers and only covers all valuations of $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ that make $d_{p+l} \equiv 1$. Thus, ITP is the decoder's Boolean function that computes $d \in \vec{d}$.

Furthermore, when $valid(\vec{f}_{p+l})$ does not 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 is not 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 three sources:

(1) Our previous paper [Shen et al. 2012].
(2) Liu et al. [Liu et al. 2012].
(3) The benchmark package sent to us by Liu, the author of [Liu et al. 2012]. So there may be some overlap between (2) and (3).

Each column of Table I shows respectively the number of input, output and register of each benchmarks. The #gate and area columns show the number of gates and area of the encoder when mapped to mcnc.genlib library by ABC synthesis tool [Berkeley Logic Synthesis and Verification Group 2008] with script "strash; dsd; strash; dc2; dc2; dch; map". In the remainder of this paper, all circuit 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 also shows how will we present the experimental result of these benchmarks:

(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 is not 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 direct generate their decoder's Boolean functions. Their experimental results will be presented in Subsection 7.5.
(3) The last two benchmarks in Table I only exist in [Liu et al. 2012]. We can not find their source code in the benchmark package sent to us by Liu. So we will not discuss them here.

We have also conducted other experiments:
In Subsection 7.6, we will compare run time overhead of two different approaches:

(1) Increasing p, l and r simultaneously in Algorithm 5 and then minimize them 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, circuit areas and delay between the two approaches that do and do not minimize p,l and r by Algorithm 6.

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 to have lots of 01 flipping	
Benchmarks in Liu [2012] and Liu's package	CC_3	1 / 2	3 / 5	13	convolution code with length 3	
	CC_4	1 / 2	4 / 7	19	convolution code with length 4	
Benchmarks in Liu's package but not in Liu [2012]	CC_13	1 / 2	13 / 36	105	convolution code with length 13	Presented in Subsection 7.5 but no data from Liu[2012] to be compared
	CC_14	1 / 2	14 / 28	84	convolution code with length 14	
	HfmAlp5	5 / 2	9/164	389	Huffman encoders of different size	
	HfmSkw5	5 / 2	10/203	482		
	HfmSkw6	6 / 2	12/421	1007		
	HfmJpg8	8 / 2	12/606	1430		
	HfmRnd8	8 / 2	13/1163	2742		
	HfmRnd9	9 / 2	14/2326	5606		
	lfsr12_6_4	1 / 1	12/15	31	Linear feedback shift register of different size	
	lfsr26_6_2	1 / 1	26/29	45		
	lfsr32_7_6	1 / 1	32/35	51		
Benchmarks in Liu[2012] but not in his package	HM(7,4)		3/50		Hamming encoder of different size	No source code, so not discussed in this paper
	HM(15,11)		4/144			

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

7.2. PCI Express 2.0 encoder

This encoder is compliant with the PCI Express 2.0 standard [PCI-SIG 2009]. After deleting empty line and comments, its source code has 259 lines of verilog.

The list of input and output variables is shown in Table II. According to the 8b/10b encoding scheme's coding table[Widmer and Franaszek 1983], when $TXDATAK \equiv 0$, $TXDATA$ can be of 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 that are not in this coding table. This assertion is embed into the transition function T, so that it can be enforced at every step in the unrolled state sequences.

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, 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

Algorithm 2 identifies the flow control variable $\vec{f} := CNTL\_TXEnable\_P0$ in 0.475 seconds. And then Algorithm 5 infers the predicate $valid(\vec{f}) := CNTL\_TXEnable\_P0$ that enables the data vector to be uniquely determined in 1.22 seconds. And then Algorithm 6 obtains the minimized $p := 3$, $l := 0$ and $r := 2$ in 0.69 seconds. Finally, with the inferred $valid(\vec{f})$, generating the decoder's Boolean functions for $CNTL\_TXEnable\_P0$, $TXDATA$ and $TXDATAK$ costs 0.26 seconds. The decoder contains 159 gates and 0 registers with area 374 and delay 9.1.

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 XGXS is compliant with clause 48 of IEEE 802.3 standard [IEEE 2012]. After deleting empty line and comments, this encoder has 214 lines of verilog.

The list of input and output variables is 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 are not in this table and embed it into the the transition function T.

Algorithm 2 identifies the flow control variable $\vec{f} := bad\_code$ in 0.31 seconds. And then Algorithm 5 infers the predicate $valid(\vec{f}) := bad\_code$ that enables the data vector to be uniquely determined in 0.95 seconds. And then Algorithm 6 obtains the minimized $p := 3$, $l := 0$ 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 138 gates and 0 registers with area 322 and delay 7.0.

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 does not 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 value of 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, 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 designed by Sun Microsystems. It is compliant with clause 36 of IEEE 802.3 standard [IEEE 2012]. After deleting empty line and comments, this encoder's source code has 864 lines of verilog.

The list of input and output variables is shown in Table IV. This encoder also employs an 8b/10b encoding scheme[Widmer and Franaszek 1983], but with yet another style of 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. It is obvious that 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 the flow control variables $\vec{f} := \{tx\_enc\_ctrl\_sel, tx\_en, tx\_er\}$ in 3.76 seconds. And then Algorithm 5 infers the predicate $valid(\vec{f}) := tx\_enc\_ctrl\_sel \equiv `PCS\_ENC\_DATA$ that enables the data vector to be uniquely determined in 21.53 seconds. And then Algorithm 6 obtains the minimized $p := 4$,$l := 0$ 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 433 gates and 9 registers with area 1020 and delay 11.5.

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 $tx\_10bdata$.

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]

<table>
<tr><th rowspan="2">Names</th><th colspan="4">Our algorithm</th><th colspan="3">[Liu et al. 2012]</th></tr>
<tr><th>time check. decoder. exist.</th><th>time gen. decoder</th><th>decoder area</th><th>decoder delay</th><th>time check. decoder exist. and gen. decoder</th><th>decoder area</th><th>decoder delay</th></tr>
<tr><td>PCIE2</td><td colspan="7" rowspan="3">Already presented in Subsection 7.2,7.3 and 7.4.</td></tr>
<tr><td>XGXS</td></tr>
<tr><td>T2Eth</td></tr>
<tr><td>XFI</td><td>13.24</td><td>6.13</td><td>7520</td><td>16.4</td><td>8.59</td><td>3913</td><td>12.5</td></tr>
<tr><td>SCRAMBLER</td><td>1.80</td><td>0.55</td><td>476</td><td>5.7</td><td>0.42</td><td>640</td><td>3.8</td></tr>
<tr><td>CC_3</td><td>0.01</td><td>0.01</td><td>6</td><td>2.0</td><td>0.21</td><td>104</td><td>9.1</td></tr>
<tr><td>CC_4</td><td>0.03</td><td>0.04</td><td>12</td><td>3.0</td><td>0.20</td><td>129</td><td>9.0</td></tr>
<tr><td>CC_13</td><td>3177</td><td colspan="4" rowspan="11">Algorithm 3 found nothing to be uniquely determined. Decoder do not exist.</td><td colspan="3" rowspan="11">no result given in [Liu et al. 2012]. So can not be compared here.</td></tr>
<tr><td>CC_14</td><td>827</td></tr>
<tr><td>HfmAlp5</td><td>0.26</td></tr>
<tr><td>HfmSkw5</td><td>0.48</td></tr>
<tr><td>HfmSkw6</td><td>1.04</td></tr>
<tr><td>HfmJpg8</td><td>1.6</td></tr>
<tr><td>HfmRnd8</td><td>9.12</td></tr>
<tr><td>HfmRnd9</td><td>56.92</td></tr>
<tr><td>lfsr12_6_4</td><td>670.79</td></tr>
<tr><td>lfsr26_6_2</td><td></td></tr>
<tr><td>lfsr32_7_6</td><td>55.47</td></tr>
</table>

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.

The first 3 benchmarks have flow control mechanism and have been discussed in Subsection 7.2,7.3 and 7.4. So they are not discussed in this subsection.

For the 2 benchmarks XFI and SCRAMBLER, 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]. 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. The other issue is that our decoder is much more larger and slower than that of [Liu et al. 2012], I think this may caused by the immature implementation of our Craig interpolant generating algorithm, which manipulate the complex proof structure generated from MiniSat in OCaml. This can be significantly improved by porting similar codes from other formal tools, such as ABC [Berkeley Logic Synthesis and Verification Group 2008].

The 2 benchmarks CC_3 and CC_4 have experimental result in [Liu et al. 2012], and also have source code in the package sent to us by its author. So we can also compare the result of our algorithm to that of [Liu et al. 2012]. But by comparing the 4th and 5th column to the 7th and 8th column, we find that our decoder area and delay is one magnitude order smaller than [Liu et al. 2012]. So I think the CC_3 and CC_4 in the package sent by Liu is not the one he used in [Liu et al. 2012]. This is further witnessed by referring to Table II of [Liu et al. 2012], which says that CC_3 and CC_4 respectively contain 6 and 7 registers, while our CC_3 and CC_4 contain respectively 3 and 4 registers. We have contacted Liu but still have not get any reply on this.

The last 11 benchmarks from CC_13 to lfsr32_7_6 have no result given in [Liu et al. 2012]. So we can only show the result of our algorithms. We have also show in Table

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

benchmarks	A1:Increasing simultaneously					A2:Increasing separately				
	p,l,r	time find. $\vec{f}$	time infer. $valid(\vec{f})$	time minimiz. p,l,r	total run time	p,l,r	time find. $\vec{f}$	time infer. $valid(\vec{f})$	time minimiz. p,l,r	total run time
PCIE2	3,0,2	0.49	1.21	0.68	2.38	3,0,2	0.38	0.80	0.38	1.60
XGXS	3,0,1	0.31	0.88	0.52	1.71	3,0,1	0.23	0.58	0.30	1.11
T2Eth	4,0,4	4.28	15.17	6.25	25.70	4,0,4	15.47	13.85	6.19	35.51
XFI	2,1,0	4.59	3.60	9.55	17.74	2,1,0	3.52	2.75	10.05	16.32
SCRAMBLER	2,1,0	0.64	0.58	1.33	2.55	2,1,0	0.48	0.43	1.47	2.38
CC_3	2,0,1	0.003	0.003	0.007	0.01	3,0,1	0.001	0.001	0.006	0.01
CC_4	2,1,1	0.005	0.004	0.011	0.02	3,1,1	0.003	0.003	0.011	0.02
CC_13	6,6,6	3177	Decoder does not exist. So no these value			6,5,6	2848.07	Decoder does not exist. So no these value		
CC_14	6,6,6	827				6,5,6	838.56			
HfmAlp5	3,3,3	0.26				3,2,2	0.15			
HfmSkw5	3,3,3	0.48				3,3,3	0.42			
HfmSkw6	3,3,3	1.04				3,3,3	1.02			
HfmJpg8	3,3,3	1.6				3,2,2	1.09			
HfmRnd8	4,4,4	9.12				4,4,4	21.25			
HfmRnd9	5,5,5	56.92				5,5,4	85.36			
lfsr12_6_4	32,32,32	670.79				32,32,32	2058.65			
lfsr26_6_2										
lfsr32_7_6	16,16,16	55.47				16,16,16	138.80			

VI that these benchmarks fail in checking their decoders' existence. So I think this is the reason that Liu have not include them in his paper [Liu et al. 2012].

7.6. Comparing run time overhead of increasing p, l and r simultaneously and separately

In Algorithm 3, we increase p, l and r simultaneously, and then reduce them with Algorithm 6. We call it A1 in this subsection.

Subsection 5.2 shows another possible way to do this. It use three nested loops to increase p, l and r separately. We call it A2 in this subsection.

We compare these two approaches in Table VII.

By comparing the total run time in column 6 and 11, it is obvious that A2 is faster than A1 in most of the case. The only exception is the T2Eth.

So does this means that we should use A2 instead of A1? The answer is NO.

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 do not have too much difference on this overhead. But for benchmarks with larger n, such as T2Eth, their difference is significant. This can be witnessed by comparing column 3 and 8, especially the row of T2Eth.

So A2 beats A1 in smaller circuits, while A1 win on larger ones. So we chose A1, that is, first increase p, l and r simultaneously, and then reduce them with Algorithm 6.

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

To improve the decoder's circuit area and timing, Algorithm 6 is invoked to reduce l and r before characterizing the decoder's Boolean function. To show its efficiency, we present the experimental result in Table VIII.

The first column is the benchmarks. When Algorithm 6 is not used, The 2nd to 6th columns give respectively the p,l and r valuation, the runtime to generate the decoder, the decoder area, the number of registers in the generated decoder, and the maximal logic delay of the decoder. When Algorithm 6 is used, these experimental result are again presented in the last 5 columns. While the 7th column presents the time used to minimize l and r.

Table VIII. Comparing run times, circuit 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	decoder area	# of reg	max logic delay	time minimiz. p,l,r	p,l,r	runtime generat. decoder	decoder area	# of reg	max logic delay
PCIE2	3,3,3	0.44	382	11	7.5	0.68	3,0,2	0.28	374	0	9.1
XGXS	3,3,3	0.35	351	20	8.2	0.52	3,0,1	0.18	322	0	7.0
T2Eth	4,4,4	4.76	1178	9	10.9	6.25	4,0,4	3.41	1557	9	11.8
XFI	2,2,2	10.67	5079	190	16.50	9.55	2,1,0	6.05	6023	58	15.5
SCRAMBLER	2,2,2	1.27	826	186	3.8	1.33	2,1,0	0.55	698	58	3.8
CC_3	Decoder do not exist. no result here										
CC_4											
CC_13											
CC_14											
HfmAlp5											
HfmSkw5											
HfmSkw6											
HfmJpg8											
HfmRnd8											
HfmRnd9											
lfsr12_6_4											
lfsr26_6_2											
lfsr32_7_6											

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.

7.8. Comparing circuit area and timing for the decoders generated by our algorithm and hand written decoders

Table IX compares the circuit area and timing of the decoders generated by our algorithm and hand written decoders.

Table IX. Comparing circuit area and timing for the decoders generated by our algorithm and hand written decoders

bench-marks	Decoder generated by us		Hand written decoder	
	area	max logic delay	area	max logic delay
PCIE2	374	9.1		
XGXS	322	7.0		
T2Eth	1557	11.8		
XFI	6023	15.5		
SCRAMBLER	698	3.8		
CC_3	Decoder do not exist. No result here			
CC_4				
CC_13				
CC_14				
HfmAlp5				
HfmSkw5				
HfmSkw6				
HfmJpg8				
HfmRnd8				
HfmRnd9				
lfsr12_6_4				
lfsr26_6_2				
lfsr32_7_6				

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 Craig 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 in Section 4, 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, which make it possible to handle encoder with flow control mechanism.

A break-through algorithm is proposed by [Tu and Jiang 2013] based on property directed reachability analysis [Bradley 2011; Eén et al. 2011] that can take the encoder's initial state into consideration, so that the infinite history of the encoder and the decoder can be used to generate the decoder's output. This algorithm can handle some special encoders that cannot be handled by the state-of-the-art algorithms. Their work is orthogonal to ours.

8.2. Program inversion

According to [Gulwani 2010], program inversion involves deriving a program P^{-1} that negates the computation of a given program P. So, the definition of program inversion is very similar to complementary synthesis.

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

Glück et al. proposed in [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.

In [Srivastava et al. 2011], Srivastava et al. assumed that an inverse program was typically related to the original program, and 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 cannot fulfill the requirement of inversion to narrow down the space of candidate programs until only the valid ones remain. So, it can only guarantee the existence of a solution, but not the correctness of this solution if its assumptions do not hold.

8.3. Protocol converter synthesis

Protocol converter synthesis is a process that automatically generates a translator between two different communication protocols. This is relevant to our work, because both focus on synthesizing communication circuits.

In [Avnit et al. 2008; Avnit et al. 2009], Avnit et al. first defined a general model for describing different protocols, and then provided an algorithm to decide whether there is some functionality of a protocol that cannot be translated into another. Finally, they synthesized a translator by computing the greatest fixed point for the update function

of the buffer's control states. Latter in [Avnit and Sowmya 2009], they improved their algorithm with a more efficient design space exploration algorithm.

8.4. Satisfying Assignments Enumeration

Some algorithms enumerate all satisfying assignments by trying to enlarge the complete satisfying assignments, so that a large state set that contains more complete satisfying assignments can be obtained.

The first approach of this kind is proposed by [McMillan 2002]. He constructs an alternative implication graph in SAT solver, which records the reasoning relation that leads to the assignment of a particular object 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 not make $obj \equiv 0$ satisfiable are removed one by one. In [Shen et al. 2005] and [Jin and Somenzi: 2005; Jin et al. 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. Cofactoring [Ganai et al. 2004] qualifies out unimportant variables by setting them to constant value returned by the SAT solver.

Other algorithms tries to construct an Interpolation to cover all satisfying assignments. The first such algorithm was proposed by [Jie-Hong Roland Jiang 2009]. It construct a first formula that contradicts with another formula to get an unsatisfiable formula, from which an interpolation can be derived and used as an over-approximation of the first formula. In [Chockler et al. 2012], interpolation is generated with a framework similar to the iterative enumerating and enlarging approaches mentioned above. But there are two enlarging steps, each for the two formulas involving in computing interpolation. This make it the first paper that constructs interpolation without proof.

8.5. Logic synthesis with Craig interpolation

In [Lee et al. 2007; Lee et al. 2008], the functional dependency and logic decomposition problems are solved by formulating the base Boolean functions' output bits as the input bits to an unknown Boolean function, and characterize this unknown function by Craig interpolation. This algorithm is also used in our paper [Shen et al. 2012] to find out all the possible decoders.

In [Wu et al. 2010], an ECO is generated with Craig interpolation.

In [Jie-Hong Roland Jiang 2009], the first algorithm to characterize a Boolean function from a Boolean Relation was proposed. It includes two different algorithms: The first one handle a general non-deterministic Boolean relation that can not uniquely determined its output, The second one is a special case of the first one that handles a deterministic relation that can uniquely determine its output by Craig interpolation. The second one is used in [Shen et al. 2012].

This paper also need to handle a non-deterministic Boolean relation, which seems to be similar to that one handled by the first algorithm of [Jie-Hong Roland Jiang 2009]. But our case is much more complicated, because the Boolean relation to be handled is an unrolled transition function with unknown length. That is, we must first find out the value of p, l and r. But these value must be determine together with finding out the set of flow control variables. So the way we handle non-determinism is significantly different from that of [Jie-Hong Roland Jiang 2009]. But after we got the value of p, l and r, together with the flow control variables $\vec{f}$ and the predicate $valid(\vec{f})$, we can

characterize the decoder's Boolean function with an algorithm similar to the second one in [Jie-Hong Roland Jiang 2009].

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

Karin Avnit, Vijay D'Silva, Arcot Sowmya, S. Ramesh, and Sri Parameswaran. 2009. Provably correct on-chip communication: A formal approach to automatic protocol converter synthesis. *ACM Transactions on Design Automation of Electronic Systems* 14, 2 (March 2009), 14:1–14:41. DOI:http://dx.doi.org/10.1145/1497561.1497562

Karin Avnit and Arcot Sowmya. 2009. A formal approach to design space exploration of protocol converters. In *Proceedings of the Conference on Design, Automation and Test in Europe,DATE 2009 (DATE '09)*. European Design and Automation Association, 3001 Leuven, Belgium, Belgium, 129–134. DOI:http://dx.doi.org/10.1109/DATE.2009.5090645

Karin Avnit, Vijay D'Silvaand Arcot Sowmya, and S. Rameshand Sri Parameswaran. 2008. A Formal Approach To The Protocol Converter Problem. In *Proceedings of the conference on Design, automation and test in Europe,DATE 2008 (DATE '08)*. ACM Press, Munich, Germany, 294–299. DOI:http://dx.doi.org/10.1109/DATE.2008.4484695

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

Aaron R. Bradley. 2011. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation, 12th International Conference, VMCAI 2011*, David A. Schmidt Ranjit Jhala (Ed.). Lecture Notes in Computer Science, Vol. 6538. Springer-Verlag, Berlin Heidelberg, 70–87. DOI:http://dx.doi.org/10.1007/978-3-642-18275-4_7

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

Sumit Gulwani. 2010. Dimensions in program synthesis. In *Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming,PPDP 2010 (PPDP '10)*. ACM Press, Hagenberg, Austria, 13–24. DOI: http://dx.doi.org/10.1145/1836089.1836091

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

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

HoonSang Jin, HyoJung Han, and Fabio Somenzi. 2005. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit. In *Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005*, Lenore D. Zuck Nicolas Halbwachs (Ed.). Lecture Notes in Computer Science, Vol. 3440. Springer-Verlag, Berlin Heidelberg, 287–300. DOI: http://dx.doi.org/10.1007/978-3-540-31980-1_19

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

Chih-Chun Lee, Jie-Hong Roland Jiang, Chung-Yang Huang, and Alan Mishchenko. 2007. Scalable exploration of functional dependency by interpolation and incremental SAT solving. In *Proceedings of 2007 International Conference on Computer-Aided Design (ICCAD '07)*. IEEE, 227–233.

Ruei-Rung Lee, Jie-Hong Roland Jiang, and Wei-Lun Hung. 2008. Bi-decomposing large Boolean functions via interpolation and satisfiability solving. In *Proceedings of the 45th Design Automation Conference, DAC 2008 (DAC '08)*. IEEE, 636–641. DOI: http://dx.doi.org/10.1145/1391469.1391634

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

Bo-Han Wu, Chun-Ju Yanga, Chung-Yang Huang, and Jie-Hong Roland Jiang. 2010. A robust functional ECO engine by SAT proof minimization and interpolation techniques. In *Proceedings of 2010 International Conference on Computer-Aided Design (ICCAD '10)*. ACM, 729–734. DOI:http://dx.doi.org/10.1109/ICCAD.2010.5654265

Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. 2001. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proceedings of the 2001 International Conference on Computer-Aided Design, ICCAD 2001 (ICCAD '01)*. ACM, 279–285.

Received February 2014; revised March 2014; accepted June 2014

Online Appendix to: Complementary Synthesis for Encoder with Flow Control Mechanism

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

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