Towards Completely Automatic Decoder Synthesis

Hsiou-Yuan Liu[1], Yen-Cheng Chou[1], Chen-Hsuan Lin[2], and Jie-Hong R. Jiang[1,2]

[1]Department of Electrical Engineering; [2]Graduate Institute of Electronics Engineering
National Taiwan University, Taipei 10617, Taiwan
jhjiang@cc.ee.ntu.edu.tw

ABSTRACT

Upon receiving the output sequence streaming from a sequential encoder, a decoder reconstructs the corresponding input sequence that streamed to the encoder. Such an encoding and decoding scheme is commonly encountered in communication, cryptography, signal processing, and other applications. Given an encoder specification, decoder design can be error-prone and time consuming. Its automation may help designers improve productivity and justify encoder correctness. Though recent advances showed promising progress, there is still no complete method that decides whether a decoder exists for a finite state transition system. The quest for completely automatic decoder synthesis remains. This paper presents a complete and practical approach to automating decoder synthesis via incremental SAT solving and Craig interpolation. Experiments show that, for decoder-existent cases, our method synthesizes decoders effectively; for decoder-nonexistent cases, our method concludes the nonexistence instantly while prior methods may fail.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design Aids—*automatic synthesis*

General Terms

algorithms, logic synthesis, verification

Keywords

Craig interpolation, decoder, finite state transition system, SAT solving

1. INTRODUCTION

Encoding and decoding processes are the cornerstone of information processing in digital communication, cryptography, digital signal processing, fault tolerant computing, and various other applications. Depending on the application, the characteristic of an encoding/decoding scheme varies. Coding systems can be designed so as to, for example, detect or correct errors in reliable communication [11], to make messages unintelligible in cryptography [16], to compress information in data processing and storage [12], to be resilient to soft errors in chip design [1, 13], and so on. Despite the diversity, encoding/decoding systems can often be modelled as finite state machines[1], e.g., convolutional codes in error correction, line codes in Ethernet and RFID, stream ciphers in symmetric encryption, etc. This paper considers the following encoding/decoding scheme. The encoder receives an input sequence and produces an output sequence; the decoder re-derives the input sequence by length-bounded partial observation of the output sequence.

As a decoder is usually harder to design than its corresponding encoder due to the fact that additional features (such as error correction) may need to be imposed, decoder design can be error prone and time consuming. Automating the process of decoder design may substantially reduce design cycle and improve circuit designers' productivity. Even if an automatically synthesized decoder would not match the same quality as a manual design, it could still be useful to justify whether the encoder is properly specified and to check if the manually crafted decoder is functionally correct. These reasons strongly motivate the study of automatic decoder synthesis.

Recently Shen et al. [14, 15] studied the decoder synthesis problem. A *bounded* decoder existence checking method was proposed [14], where the checking is with respect to a pre-specified parameter on observable output windows. If a decoder exists, an ALLSAT-based procedure is invoked to compute and simplify the corresponding decoding functions. The necessity of pre-specifying the checking bound prevents decoder synthesis from being a fully automatic process. A later attempt [15] got one step closer to *unbounded* decoder existence checking. Despite its soundness, the proposed checking is unfortunately incomplete. Essentially there are cases that the checking never halts, in particular, when a decoder does not exist. Figure 1 shows one such example, where a decoder does not exist, but the checking fails to decide.[2] Nevertheless, the approach works well on practical design instances.

This paper continues the quest for a sound and complete approach to automatic decoder synthesis. The main advances include the following results: Firstly, a necessary and sufficient condition for decoder existence is identified. Secondly, a complete decoder existence checking procedure is proposed with guaranteed termination within $O(N^2)$ iterations, where N is the number of states of a state transition system. Thirdly, an interpolation-based decoder synthesis approach is proposed, which eliminates the need for ALLSAT in enumerating all satisfying assignments and makes a

[1]Memoryless (or combinational) encoding/decoding can be thought of as a single-state finite state machine.

[2]The problem results from the misconception that the notion of *unique states* [15] exactly captures the essence of decoder existence. However, there are state transition systems that consist of purely unique states and yet have no decoder as the example of Figure 1 suggests.

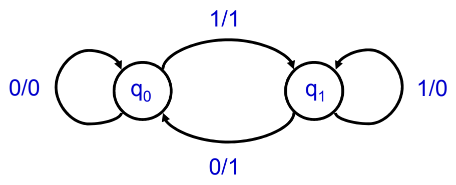

Figure 1: A 0-1 alternation detector with unspecified initial states.

decoder derivable along the existence checking. (Shen et al. [14] suggested as future work using interpolation-based relation determinization [7] for decoder generation. Our interpolation formulation for decoder synthesis can be more direct and simpler than the prior method [7].) Finally, two techniques, CNF encoding of disjunctive constraints and incremental time-frame expansion with reused looping constraints, are proposed to enhance the efficiency of incremental SAT solving. Experiments show that our algorithm successfully decides decoder existence, while the prior method may fail, and effectively synthesizes decoders if they do exist.

This paper is organized as follows. Section 2 gives the preliminaries. Our main results on decoder existence checking and synthesis are presented in Section 3. Implementation issues are discussed in Section 4. The proposed methods are evaluated with experimental results in Section 5. Finally Section 6 concludes this paper.

2. PRELIMINARIES

As conventional notation, the cardinality of a vector $\vec{x} = (x_1, \ldots, x_k)$ is denoted as $|\vec{x}| = k$. For $\vec{x}$ being a vector of Boolean variables, its set of truth valuations is denoted $[\![\vec{x}]\!]$, e.g., $[\![(x_1, x_2)]\!] = \{(0,0), (0,1), (1,0), (1,1)\}$.

Let $V = \{v_1, \ldots, v_k\}$ be a finite set of Boolean variables. A *literal* l is either a Boolean variable v_i or its negation $\neg v_i$. A *clause* C is a disjunction of literals. A conjunction of clauses is in the so-called *conjunctive normal form* (CNF). In the sequel, a clause set $S = \{C_1, \ldots, C_k\}$ shall mean to be the CNF formula $C_1 \wedge \cdots \wedge C_k$. An *assignment* over V gives every variable v_i a Boolean value either 0 or 1. A CNF formula is *satisfiable* if there exists a satisfying assignment such that the formula evaluates to 1. Otherwise it is *unsatisfiable*.

2.1 SAT Solving and Craig Interpolation

We assume the reader's familiarity with *satisfiability* (SAT) solving, Craig interpolation, and circuit-to-CNF conversion. We omit essential backgrounds and refer the reader to [8].

To introduce terminology and convention for later use, we restate the following theorem.

Theorem 1 (Craig Interpolation Theorem). *[4] For two Boolean formulas ϕ_A and ϕ_B, with $\phi_A \wedge \phi_B$ unsatisfiable, there exists a Boolean formula ψ_A referring only to the common variables of ϕ_A and ϕ_B such that ϕ_A implies ψ_A and $\psi_A \wedge \phi_B$ remains unsatisfiable.*

The Boolean formula ψ_A is referred to as the *interpolant* of ϕ_A with respect to ϕ_B. We shall assume that ϕ_A and ϕ_B are in CNF. So a refutation proof of $\phi_A \wedge \phi_B$ is available from a SAT solver. Further, an interpolant circuit ψ_A can be constructed from the refutation proof in linear time [9].

2.2 State Transition Systems

We model a synchronous sequential circuit as a *(finite state) transition system* in terms of two characteristic functions $I(\vec{s})$, representing the initial states, and $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}\,')$, representing the transition relation, where $\vec{s}$, $\vec{s}\,'$, $\vec{x}$, and $\vec{y}$ are referred to as the current-state variables, next-state variables, input variables, and output variables, respectively. In the sequel, we shall specify a transition system with its transition relation only when its initial states are immaterial. Moreover, as we are concerned about deterministic systems, we sometimes abuse the relation notation to mean the transition function $T : [\![\vec{x}]\!] \times [\![\vec{s}]\!] \rightarrow [\![\vec{y}]\!] \times [\![\vec{s}\,']\!]$.

For a state transition system T, we distinguish three types of states: First, a *dangling state* is a state without predecessors or, recursively, a state with only dangling predecessors. (For a state pair ($\vec{q} \in [\![\vec{s}]\!], \vec{q}\,' \in [\![\vec{s}\,']\!]$) satisfying $\exists \vec{x}, \exists \vec{y}. T(\vec{x}, \vec{q}, \vec{y}, \vec{q}\,')$, we call $\vec{q}$ the *predecessor* of $\vec{q}\,'$.) Second, a *recurrent states* is a state that can reach itself within a finite number of transition steps. (So a recurrent state must be non-dangling.) Third, a *transient state* is a non-dangling state not in any loop. (Therefore these three types form a partition on the state space of T.)

For decoder synthesis to be discussed, we apply time-frame expansion on a transition system $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}\,')$, similar to bounded model checking [2]. In the sequel, the variable vector $\vec{v}$ instantiated at time-frame t shall be denoted as $\vec{v}^{\,t}$. With a slight extension, the transition relation unrolled at time t shall be denoted as T^t to mean $T(\vec{x}^{\,t}, \vec{s}^{\,t}, \vec{y}^{\,t}, \vec{s}^{\,t+1})$, where t can be positive or negative with respect to a reference time point at $t = 0$. Similarly, we let T^* denote the transition relation the same as T except that variables $\vec{x}$, $\vec{s}$, $\vec{y}$, and $\vec{s}\,'$ of T are substituted with fresh new variables $\vec{x^*}$, $\vec{s^*}$, $\vec{y^*}$, and $\vec{s^*}'$, respectively.

2.3 Problem Statement

Given an encoder in the form of a state transition system T, which transforms an input sequence to an output sequence according to the transition relation, the decoder to-be-synthesized aims to reproduce the input sequence by observing the output sequence.

For a decoder to be realizable, we shall base assumptions on the following facts. Firstly, since the lengths of input and output sequences can be unbounded, decoding must be done online (processing data piece-by-piece serially) rather than offline (processing entire data at once). Secondly, since the decoder should have only finite memory, the input value at a time point should be decided upon observing only a finite portion of the output sequence. Thirdly, in general the input sequence cannot be recovered since the very first input value because, to determine the input value at time t, some output values before t need to be known. Therefore a certain delay may be necessary before an input value can be uniquely determined. In certain applications (such as communicating and reactive systems) losing first few input values is immaterial. A decoder may or may not recover a certain prefix of an input sequence depending on whether or not past output values are needed.

For decoder synthesis, only the reachable non-dangling states of a transition system T are of our interests. Given an exact or over-approximated care-state set S_C, it can be exploited to accelerate decoder existence checking and improve decoder synthesis. (The care-state set S_C can be generated by exact or approximated reachability analysis. For example, the latter approach was taken in [14] by time-frame expansion for dangling-state removal.) In the sequel, we shall simply assume that a care-state set S_C is given. Moreover, we shall not distinguish a characteristic function and the set that it represents. (When care states are not known, we treat all states as care states, thus having characteristic function $S_C(\vec{s}) = 1$.) Similar to the conventions T^t and T^* of T, we let S_C^t mean $S_C(\vec{s}^{\,t})$ and S_C^* mean $S_C(\vec{s^*})$.

Another source of don't cares comes from inputs. Often we are only interested in decoding a design under its certain operation modes. This paper assumes a transition system has been constrained to its proper operation modes from its original design.

3. MAIN ALGORITHMS

3.1 Decoder Existence

The necessary and sufficient condition for decoder existence with respect to a pre-specified observation constraint can be stated as follows.

THEOREM 2 (SEE ALSO [14]). *Given a transition system $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$, suppose that to determine is the input $\vec{i}^0 \in [\![\vec{x}^0]\!]$ at some relative reference time point of $t = 0$ by observing the outputs $\vec{o}^t \in [\![\vec{y}^t]\!]$ at time $t = -n, \dots, p$ for $n, p \geq 0$. Input $\vec{i}^0$ can be uniquely determined from $\vec{o}^{-n}, \dots, \vec{o}^p$ if and only if the formula*

$$\bigwedge_{t=-n}^{p} \left(T^t \wedge T^{*t} \wedge (\vec{y}^t = \vec{y^*}^t)\right) \wedge (\vec{x}^0 \neq \vec{x^*}^0) \wedge \bigwedge_{t=-n}^{p+1} \left(S_C^t \wedge {S_C^*}^t\right), \quad (1)$$

is unsatisfiable, where predicate "=" asserts the bit-wise equivalence of its two argument variable vectors and "≠" asserts the negation.

Intuitively, the input sequence of a state transition system can be reverse engineered if the input at some time point can be uniquely determined from its proximate output string. In essence, the parameter (n, p) defines an *observation window* on the output sequence for decoder synthesis. By sliding the window along an output sequence, the original input sequence can be recovered. (When n is non-zero, the first n values of an input sequence cannot be determined. Hence in decoder synthesis it is desirable for n to be small.) For simplicity, unless otherwise said we shall assume that $S_C(\vec{s}) = 1$ in the sequel.

Formula (1) can be visualized as the circuit construction shown in Figure 2(a), where T is meant to be the transition function instead of relation. In the sequel, we call it the (n, p)-*miter*, denoted $M(n, p)$, of transition system T from the $-n$th to pth time-frame. Hence $M(n, p)$ *equally denotes Formula (1).*

Notice that Formula (1) tests decoder existence only with respect to a pre-specified n, p parameter. Its satisfiability yields no conclusive answer whether the decoder does not exist at all or the decoder exists at some larger n, p. When there is no decoder at all, the test for even larger n, p may continue forever. A terminate condition must be imposed to prevent infinite trials.

The following lemma asserts the necessary and sufficient condition for decoder existence.

LEMMA 1. *The decoder of a transition system $T(\vec{s}, \vec{x}, \vec{s}', \vec{y})$ does not exist if and only if there exist two distinct inputs $\vec{i_1}^0, \vec{i_2}^0 \in [\![\vec{x}^0]\!]$ at time $t = 0$ that are consistent (in terms of input-output traces) with some same infinite output sequence*

$$\dots, \vec{o}^{-1}, \vec{o}^0, \vec{o}^1, \dots,$$

for $\vec{o}^t \in [\![\vec{y}^t]\!]$, constrained by the transition relation T.

PROOF. (⇐) The encoder input cannot be uniquely determined by output sequences of bounded lengths as this infinite output sequence provides a counterexample.

(⇒) Consider the contrapositive. For every pair of distinct inputs $\vec{i_1}^0$ and $\vec{i_2}^0$, any output sequence consistent with both $\vec{i_1}^0$ and $\vec{i_2}^0$ is bounded from below or above. Because T is of finite states, as long as the output sequence is long enough a state pair $(\vec{q_1}, \vec{q_2}) \in [\![\vec{s}]\!] \times [\![\vec{s}]\!]$, for $\vec{q_1}$ and $\vec{q_2}$ on the state traces consistent with $\vec{i_1}^0$ and $\vec{i_2}^0$, respectively, will eventually repeat. This repetition makes the output sequences unboundedly extendable. Therefore, for those output sequences bounded from below (above), there exists a global lower bound $l \leq 0$ (upper bound $u \geq 0$) such that none of them starts before $t = l$ (ends after $t = u$). Let l^* and u^* be the minimum lower bound and maximum upper bound, respectively, among all distinct input pairs $\vec{i_1}^0$ and $\vec{i_2}^0$. By observing any output sequence with $t = l^* - 1, \dots, u^* + 1$, its corresponding input at $t = 0$ is unique. Thus the decoder of T exists. ■

It is important to notice that the infinity of the output sequence must go in both positive and negative directions. A decoder exists if *every* output sequence consistent with two distinct inputs $\vec{i_1}^0, \vec{i_2}^0$, if unbounded in length, extends to infinity in only one direction.

Based on Lemma 1, the following theorem lays the computational foundation for decoder existence checking.

THEOREM 3. *The decoder of a transition system $T(\vec{s}, \vec{x}, \vec{s}', \vec{y})$ does not exist if and only if the formula*

$$M(n,p) \wedge \left(L^{\pm}_{n,p} \vee (L^-_n \wedge L^+_p)\right), \quad (2)$$

where

$$L^{\pm}_{n,p} = \bigvee_{i=-n}^{0} \bigvee_{j=1}^{p+1} \left((\vec{s}^i = \vec{s}^j) \wedge (\vec{s^*}^i = \vec{s^*}^j)\right), \quad (3)$$

$$L^-_n = \bigvee_{i=-n}^{-1} \bigvee_{j=i+1}^{0} \left((\vec{s}^i = \vec{s}^j) \wedge (\vec{s^*}^i = \vec{s^*}^j)\right), \text{ and} \quad (4)$$

$$L^+_p = \bigvee_{i=1}^{p} \bigvee_{j=i+1}^{p+1} \left((\vec{s}^i = \vec{s}^j) \wedge (\vec{s^*}^i = \vec{s^*}^j)\right), \quad (5)$$

is satisfiable under some n, p. (L^-_n and L^+_p are defined to be false for $n = 0$ and $p = 0$, respectively.)

PROOF. Consider $T \wedge T^*$ as the product transition system of T and T^*. It induces state transitions in the product state space $[\![\vec{s}]\!] \times [\![\vec{s^*}]\!]$.

(⇐) The satisfiability of Formula (2) under some n, p indicates $M(n, p) \wedge L^{\pm}_{n,p}$ or $M(n, p) \wedge L^-_n \wedge L^+_p$ is satisfiable. Let $(\vec{q}^0, \vec{q^*}^0)$ be a satisfying state at time $t = 0$. The former suggests $(\vec{q}^0, \vec{q^*}^0)$ is in a loop of the product transition system $T \wedge T^*$. As a consequence, a satisfying output sequence $\vec{o}^{-n} = \vec{o^*}^{-n}, \dots, \vec{o}^p = \vec{o^*}^p$ can be infinitely extended in both positive and negative directions. By Lemma 1, the decoder does not exist. The latter suggests that $(\vec{q}^0, \vec{q^*}^0)$ is a state that can be reached by a loop satisfying L^-_n and can reach another loop satisfying L^+_p. Because of these two loops, a satisfying output sequence can be infinitely extended in both positive and negative directions, and thus the decoder does not exist as well.

(⇒) Consider the contrapositive. Suppose there is no n, p that make Formula (2) satisfiable. It implies that any $(\vec{q}^0, \vec{q^*}^0)$ satisfying $M(n, p)$ is neither in some loop, nor between two loops. Moreover, because $T \wedge T^*$ is a *finite state* transition system, any output sequence satisfying $M(n, p)$ cannot be infinitely extended to both positive and negative directions. By Lemma 1, a decoder must exist. ■

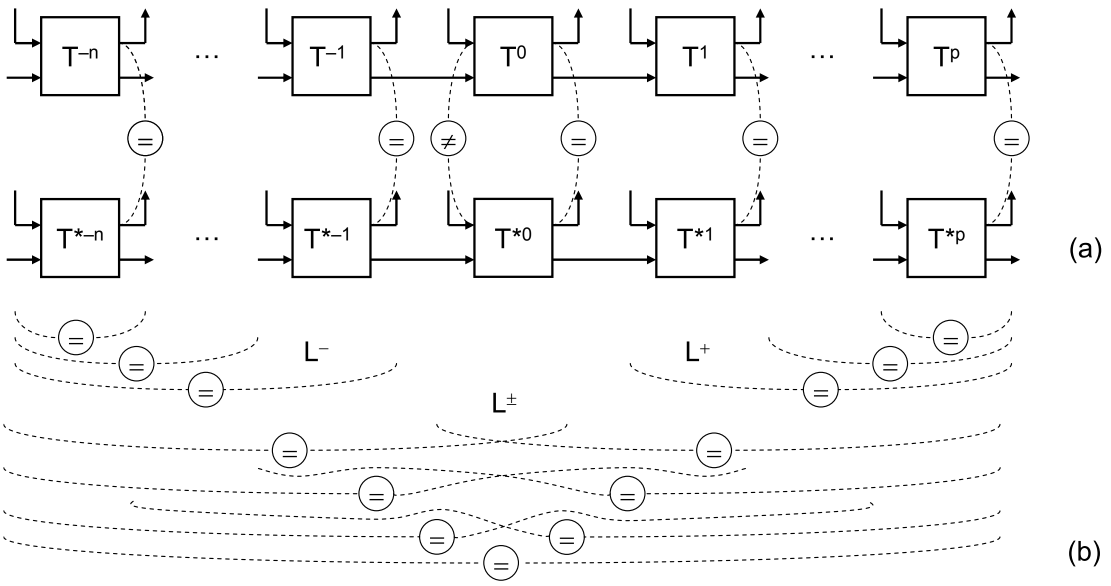

Figure 2: (a) Decoder (n, p)-miter of transition system T; (b) looping constraints L^-, L^+, and $L^\pm$ for the state variables of the (n, p)-miter in (a).

Note that the looping constraint $L^\pm_{n,p}$ of Formula (2) is not essential. If $M(n,p) \wedge L^\pm_{n,p}$ is satisfiable, then there must exist some $n' \geq n$ and $p' \geq p$ making $M(n', p') \wedge L^-_{n'} \wedge L^+_{p'}$ satisfiable. This constraint however can be useful in shortening the witnessed counterexample to decoder existence. On the contrary, $L^-_n \wedge L^+_p$ is irreplaceable by $L^\pm_{n',p'}$ for some n', p' because the state $(\vec{q}^{\,0}, \vec{q^*}^{\,0}) \in [\![\vec{s}^{\,0}]\!] \times [\![\vec{s^*}^{\,0}]\!]$ satisfying $M(n,p) \wedge L^-_n \wedge L^+_p$ can be a transient state between two loops rather than in a loop.

By Theorems 2 and 3, the existence of a decoder for a given transition system T can be checked with the algorithmic flow in Figure 3. Among the three SAT solving instances of the procedure, the first and second follow from Theorems 2 and 3, respectively. The third, on the other hand, is optional. That is, if the second formula $M(n,p) \wedge (L^\pm_{n,p} \vee (L^-_n \wedge L^+_p))$ is unsatisfiable, then both n and p can directly be incremented by 1 to start a new iteration. Solving the third formula $M(n,p) \wedge (L^-_n \vee L^+_p)$, however, may result in better termination condition with smaller n and p as the following proposition suggests.

Proposition 1. *Assume that $M(n,p)$ is satisfiable but not $M(n,p) \wedge (L^\pm_{n,p} \vee (L^-_n \wedge L^+_p))$. If $M(n,p) \wedge L^-_n$ (respectively $M(n,p) \wedge L^+_p$) is satisfiable, then incrementing p (respectively n) only achieves the tightest increase on current (n,p) without missing any termination condition.*

Proof. Consider first the formula $M(n,p) \wedge L^-_n$. For $M(n,p)$ satisfiable but not $M(n,p) \wedge (L^\pm_{n,p} \vee (L^-_n \wedge L^+_p))$, then a satisfying solution to it must correspond to a valid loop in the negative time-frames while there is no valid loop in the positive time-frames. Since the truth assignments in this loop can be arbitrary extended to the negative direction, the current satisfying assignment of $M(n,p) \wedge L^-_n$ must remain valid for $M(n+1,p) \wedge L^-_{n+1}$. Moreover, for this assignment, no new loop can be created in the positive time-frames satisfying $M(n+1,p) \wedge L^+_p$ because $M(n+1,p) \wedge L^+_p \Rightarrow M(n,p) \wedge L^+_p$. Therefore incrementing n can neither exclude the current satisfying solution, nor make this assignment a counterexample. On the other hand, even if incrementing n results in satisfiable $M(n+1,p) \wedge L^\pm_{n+1,p}$, the same loop can be created $M(n,p+1) \wedge L^\pm_{n,p+1}$. Consequently, we only need to increment p. Similarly, for $M(n,p) \wedge L^+_p$, we only need to increment n. ■

The procedure of Figure 3 always terminates as the following theorem asserts.

Theorem 4. *Given a transition system $T(\vec{x}, \vec{s}, \vec{y}, \vec{s}')$ and its care-state set $S_C \subseteq [\![\vec{s}]\!]$, the decoder existence checking procedure of Figure 3 terminates with $n + p \leq |S_C|^2$.*

Proof. When no decoder exists, a counterexample must be in the form of either a loop or two connected (state-disjoint) loops in the product space of $T \wedge T^*$. In either case, the transition span of a counterexample is upper bounded by $|S_C|^2$. Hence $n + p \leq |S_C|^2$.

When a decoder exists, the unsatisfiability of $M(n,p)$ can always be established whenever the transition span of the longest loop and the transition span of the longest connected two loops have been reached, which are both upper bounded by $|S_C|^2$. Hence $n + p \leq |S_C|^2$. ■

(When $S_C(\vec{s}) = 1$, of course $n + p \leq 2^{|\vec{s}|}$.)

Corollary 1. *The procedure of Figure 3 always terminates with a correct answer.*

Upon termination, however, the corresponding (n,p) may not be minimal because in a solving iteration, when the first SAT instance is satisfiable but not the second and third, the increment of both n and p is not tight. Essentially in this case we do not know whether incrementing p only or n only leads to a better solution.

In the decoder existence checking, both n and p start from 0 and increase by 1 until either a decoder is found or its existence is falsified. This increment permits simplification to the looping constraints of Formulas (3), (4), and (5). Consider the simplification of Formula (3). Observe that n and p are simultaneously incremented only because of the unsatisfiability of $M(n,p) \wedge (L^-_n \vee L^+_p)$. Moreover, $M(n+1,p+1) \Rightarrow M(n,p)$, $L^-_n \Rightarrow L^-_{n+1}$, and $L^+_p \Rightarrow L^+_{p+1}$. Therefore the satisfiability of $M(n+1,p+1) \wedge (L^-_{n+1} \vee L^+_{p+1})$ can only be attributed to the extra equalities existing in L^-_{n+1} but not in L^-_n. Formula (3)

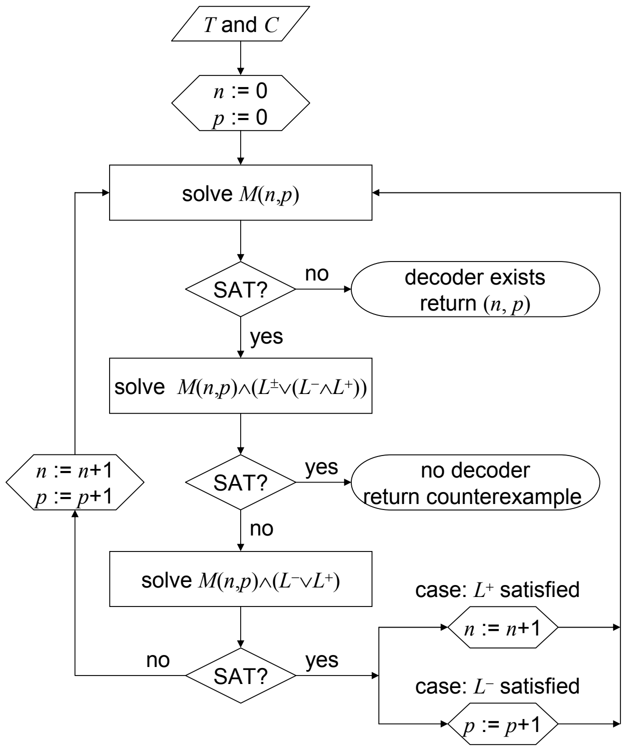

Figure 3: Decoder existence checking.

can thus be simplified to Formula (6) below. Similarly, we have Formulas (7) and (8).

$$
\begin{aligned}
L^{\pm}_{n,p} &= \bigvee_{j=1}^{p+1} \left((\vec{s}^{\,-n} = \vec{s}^{\,j}) \wedge (\vec{s^*}^{-n} = \vec{s^*}^{j}) \right) \vee \\
&\quad \bigvee_{j=-n}^{0} \left((\vec{s}^{\,p+1} = \vec{s}^{\,j}) \wedge (\vec{s^*}^{p+1} = \vec{s^*}^{j}) \right), \qquad (6) \\
L^{-}_{n} &= \bigvee_{j=-n+1}^{0} \left((\vec{s}^{\,-n} = \vec{s}^{\,j}) \wedge (\vec{s^*}^{-n} = \vec{s^*}^{j}) \right), \text{ and } (7) \\
L^{+}_{p} &= \bigvee_{j=1}^{p} \left((\vec{s}^{\,p+1} = \vec{s}^{\,j}) \wedge (\vec{s^*}^{p+1} = \vec{s^*}^{j}) \right). \qquad (8)
\end{aligned}
$$

As a result, the original quadratic numbers of equality constraints are reduced to linear. The looping constraints of Formula (2) are shown in Figure 2(b) in connection to the miter constraint $M(n,p)$ shown in Figure 2(a). The equality signs in this figure signify the equality constraints imposed on the state variables among the time-frames of $M(n,p)$.

3.2 Decoder Synthesis

When a decoder exists, we proceed synthesizing it under the (n,p) observation window returned by the above decoder existence checking procedure. The decoder can be synthesized for all bits $\vec{x}^{\,0}$ at once or for every bit $x_i^0 \in \vec{x}^{\,0}$ one at a time. For the sake of optimality, we adopt the latter strategy. By synthesizing the decoding function f_i for each bit $x_i^0 \in \vec{x}^{\,0}$, the actual necessary window, specified by (n_i, p_i) for some $0 \le p_i \le p$ and $-p_i \le n_i \le n$, can be substantially

```
DecoderSynthesis
  input: transition system $T$, care states $S_C$, parameter $(n,p)$
  output: decoding functions
  begin
  01  for $i = 1, \ldots, |\vec{x}|$
  02      search minimal $n_i$ and $p_i$ for $M_i(n,p,n_i,p_i)$ unsat
  03      derive $f_i$ by interpolation on $\phi_{i_A} \wedge \phi_{i_B}$
  04  return $(f_1, \ldots, f_{|\vec{x}|})$
  end
```

Figure 4: Algorithm: Decoder synthesis.

reduced. Specifically, the formula

$$
\bigwedge_{t=-n}^{p} \left(T^t \wedge T^{*t} \right) \wedge \bigwedge_{t=-n}^{p+1} \left(S_C^t \wedge S_C^{*\,t} \right) \wedge \bigwedge_{t=-n_i}^{p_i} \left(\vec{y}^{\,t} = \vec{y^*}^{t} \right) \wedge (x_i^0 \neq x_i^{*0}). \quad (9)
$$

denoted $M_i(n,p,n_i,p_i)$, must remain unsatisfiable as $M(n,p)$. So the corresponding decoding function f_i to be derived by interpolation from the refutation proof of $M_i(n,p,n_i,p_i)$ may have fewer support variables and a simpler circuit structure.

The validity of synthesizing one decoding function at a time stems from the following fact, provable by $M(n,p) = \bigvee_{i=1}^{|\vec{x}|} M_i(n,p,n,p)$.

Proposition 2. *For a state transition system $T(\vec{x},\vec{s},\vec{y},\vec{s}')$, $M(n,p)$ is unsatisfiable if and only if $M_i(n,p,n,p)$ is unsatisfiable for every $i = 1, \ldots, |\vec{x}|$.*

For unsatisfiable $M_i(n,p,n_i,p_i)$, Craig interpolation (Theorem 1) can be exploited to derive the decoding function f_i of x_i^0 as Theorem 5 suggests. (A similar construction using Craig interpolation has been proposed in [8] for logic synthesis application.)

Theorem 5. *For a transition system $T(\vec{x},\vec{s},\vec{y},\vec{s}')$ with unsatisfiable $M_i(n,p,n_i,p_i)$, let formulas ϕ_{i_A} and ϕ_{i_B} be*

$$
\phi_{i_A} : \bigwedge_{t=-n}^{p} T^t \wedge \bigwedge_{t=-n}^{p+1} S_C^t \wedge x_i^0, \text{ and} \qquad (10)
$$

$$
\phi_{i_B} : \bigwedge_{t=-n}^{p} T^{*t} \wedge \bigwedge_{t=-n}^{p+1} S_C^{*\,t} \wedge \bigwedge_{t=-n_i}^{p_i} \left(\vec{y}^{\,t} = \vec{y^*}^{t} \right) \wedge \neg x_i^{*0}. \qquad (11)
$$

Then the interpolant ψ_{i_A} of ϕ_{i_A} with respect to ϕ_{i_B} is a valid decoding function for $x_i^0 \in \vec{x}^{\,0}$.

Proof. Observe first that $M_i(n,p,n_i,p_i)$ and $\phi_{i_A} \wedge \phi_{i_B}$ are satisfiability equivalent. So $\phi_{i_A} \wedge \phi_{i_B}$ is unsatisfiable.

By Theorem 1, we know that the interpolant ψ_{i_A} refers only to $\vec{y}^{\,-n_i}, \ldots, \vec{y}^{\,p_i}$, the common variables of ϕ_{i_A} and ϕ_{i_B}. For $\phi_{i_A} \Rightarrow \psi_{i_A}$ by Theorem 1, any output sequence $\vec{o}^{\,-n_i} \in [\![\vec{y}^{\,-n_i}]\!]$, …, $\vec{o}^{\,p_i} \in [\![\vec{y}^{\,p_i}]\!]$ that makes ϕ_{i_A} satisfiable and thus asserts x_i^0 will be in the onset of ψ_{i_A}. For $\psi_{i_A} \wedge \phi_{i_B}$ unsatisfiable by Theorem 1, any output sequence $\vec{o}^{\,-n_i}, \ldots, \vec{o}^{\,p_i}$ that makes ϕ_B satisfiable and thus asserts $\neg x_i^{*0}$ will be in the offset of ψ_{i_A}. On the other hand, since $\phi_{i_A} \wedge \phi_{i_B}$ is unsatisfiable, there is no output sequence $\vec{o}^{\,-n_i}, \ldots, \vec{o}^{\,p_i}$ in both onset and offset of ψ_{i_A}. Hence ψ_{i_A} defines a valid decoding function for x_i^0 of $\vec{x}^{\,0}$. ■

Based on Theorem 5, the procedure of interpolation-based decoder synthesis is sketched in Figure 4.

4. IMPLEMENTATION DETAILS

We discuss two implementation issues and their solutions.

4.1 CNF Encoding of Disjunctive Constraints

The disjunctive constraints encountered in this paper are of the form $\varphi_1 \vee \cdots \vee \varphi_\ell$, where φ_i's are CNF formulas. Let φ_i consist of k_i clauses $\{C_{i1}, \ldots, C_{ik_i}\}$. Then

$$\bigvee_{i=1}^{\ell} (C_{i1} \wedge \cdots \wedge C_{ik_i}) \tag{12}$$

can be converted to CNF as

$$\bigwedge_{i=1}^{\ell} ((C_{i1} \vee \neg c_i) \wedge \cdots \wedge (C_{ik_i} \vee \neg c_i)) \wedge (c_1 \vee \cdots \vee c_\ell), \tag{13}$$

where c_i's are fresh new variables.

Proposition 3. *Formulas (12) and (13) are equisatisfiable.*

Thereby the vector inequality can be easily expressed in CNF.

In decoder existence checking, however, the disjunction list $\varphi_1 \vee \cdots \vee \varphi_\ell$ may increase over time, i.e., ℓ increases. To support incremental SAT solving, we further modify the above conversion and recursively define

$$\Phi_i = \Phi_{i-1} \wedge \bigwedge_{j=1}^{k_i} (C_{ij} \vee \neg c_i) \wedge (b_{i-1} \vee c_i \vee \neg b_i), \tag{14}$$

for $\Phi_0 = 1$ and $b_0 = 0$.

Proposition 4. *Formula (13) and formula $\Phi_\ell \wedge b_\ell$ are equisatisfiable.*

Note that, since literal b_ℓ can be asserted by *unit assumption* [5], Formula (14) is extendable to arbitrary ℓ for incremental solving. Thereby the looping constraints can be incrementally expressed in CNF.

4.2 Incremental Time-Frame Expansion

There are different strategies of inserting a new time-frame into an expanded array of time-frames. Due to the looping constraints, in the decoder existence checking procedure of Figure 3 we prefer the following insertion strategy.

For n to be incremented, a new time-frame is inserted between the 0^{th} and the -1^{st} time-frames, rather than appending before the $-n^{\text{th}}$. Effectively, the variables with original time-indices $t = -1, -2, \ldots,$ and $-n$ of Formula (2) are relabelled with $t = -2, -3, \ldots,$ and $-(n+1)$, respectively. For p to be incremented, on the other hand, a new time-frame is inserted between the 0^{th} and the 1^{st} time-frames, rather than appending after the p^{th}. Effectively, the variables with original time-indices $t = 1, 2, \ldots,$ and p are relabelled with $t = 2, 3, \ldots,$ and $p+1$, respectively. Moreover the reconnection between the new time-frame and existing time-frames can be done via proper utilization of unit assumptions.

Under this strategy, all the clauses of looping constraints added before remain in use. Only two equality constraints (i.e., $\vec{s}^{-(n+1)} = \vec{s}^0$ and $\vec{s}^{p+1} = \vec{s}^0$ for n incremented, and $\vec{s}^{-n} = \vec{s}^1$ and $\vec{s}^{p+2} = \vec{s}^1$ for p incremented) need to be added per time-frame expansion. In contrast, if we were to append a new time-frame at the end of the array, we would have to add $(n+p+2)$ looping constraints related to the new time-frame added. It results in a more complicated formula and less effective reuse of learned clauses.

5. EXPERIMENTAL RESULTS

The proposed method, named Decosy, were implemented in ABC [3]. The experiments were conducted on a Linux machine with Xeon 2.53GHz CPU and 48GB RAM. The benchmark circuits and executable codes of prior work [14, 15] were obtained online [6]. The profiles of circuits `XGXS`, `XFI`, `Scrambler`, `PCIE`, and `T2Ethernet` can be found in [14]. Two additional designs: the `HM` series, implementing the Hamming codes for correcting any 1-bit error, and `AD`, implementing the 0-1 alternation detector of Figure 1, were created. The circuits in Verilog were converted to the blif format for optimization in ABC. The final decoder circuits were mapped into standard cells with the *mcnc.genlib* library.

We conducted three sets of experiments: comparison with [14] on decoder generation in Table 1, comparison with [15] on decoder existence checking and generation in Table 2, and comparison with [15] on decoder existence checking for circuits without decoders in Table 3. Note that the executables of [14, 15] were implemented in OCaml running zChaff [10], whereas ours were implemented in C running MiniSat [5]. The reported runtimes in [14] and [15], which were obtained on different machine settings, were repeated in the parentheses in the fifth column of Table 2 and the second column of Table 3 for reference. It is interesting to notice the curious runtime inconsistencies.

Table 1 compares decoder generation results of [14] and Decosy with respect to the pre-specified parameters given in [14], which are not repeated here to save space. The obtained decoder circuits were optimized with ABC under script `strash`; `dsd`; `strash`; `dc2`; `dc2`; `dch`; `map`. The decoder area, delay, and computation time (including decoder generation time plus script optimization time in seconds) are shown. (The decoders generated by [14] were in Verilog format, and were converted to blif for optimization under the same script.) As shown, the optimization script effectively reduced all of the decoders generated by the prior method and Decosy within 2.13 seconds. Except for `PCIE` and `T2Ethernet`, Decosy achieved similar or better results. For `PCIE` and `T2Ethernet`, the xor-minimization efforts of [14] were likely taking effect (as noted in [14] that communication circuits are commonly xor-dominated). On the other hand, for the larger circuits `XFI` and `HM(15,11)`, Decosy achieved more impressive improvements. (For the `HM` circuits, the prior method missed decoder generation at the time-frame expansion where the decoder is supposed to exist, perhaps due to implementation problems. The data, marked '§' in Table 1 as well as in Table 2, were obtained by our own re-implementation of [14] for referential purposes.)

Table 2 compares the results of [15] and Decosy for decoder existence checking plus decoder generation. It lists obtained parameters $(n, p, n^\dagger, p^\dagger)$, numbers of decoder inputs/registers, circuit area/delay, and runtime, where $n^\dagger = \max_i n_i$ and $p^\dagger = \max_i p_i$ by the notation of Section 3.2. The runtime includes checking decoder existence and script optimization, same as those of Table 1.

Table 3 compares the runtime (in seconds) of [15] and Decosy for decoder nonexistence checking. Circuits `XGXS_err`, `XFI_err`, `Scrambler_err`, `PCIE_err`, and `T2Ethernet_err` are obtained via design error insertion in [14]. The `HM_err` circuits, on the other hand, were derived by embedding noisy channels with memory and multi-bit flipping capability into the `HM` circuits. These circuits and `AD` have no decoders. In all the cases Decosy concluded decoder nonexistence under parameters $(n, p) = (0, 0)$, i.e., without time-frame expansion, except for the `HM_err` series requiring multiple time-frame expansion. It tends to suggest that Decosy can be effective in detecting decoder non-existence and beneficial to assisting design verification. In contrast, the prior method [15] is incomplete and less effective.

6. CONCLUSIONS

We have presented the first sound and complete approach

Table 1: Comparison on Decoder Generation.

circuit	[14]		Decosy		area ratio	delay ratio
	area/delay	time	area/delay	time		
XGXS	269/7.4	1.23 = 1.17+0.06	286/7.3	0.08 = 0.02+0.06	1.06	0.99
XFI	5697/14.4	492.58 = 490.45+2.13	3978/14.3	4.02 = 3.21+0.81	0.70	0.99
Scrambler	736/3.8	1.88 = 1.83+0.05	640/3.8	0.25 = 0.19+0.06	0.87	1
PCIE	171/5.8	1.04 = 1.02+0.02	190/6.6	0.08 = 0.04+0.04	1.11	1.14
T2Ethernet	299/7.5	22.67 = 22.62+0.05	583/9.0	1.47 = 1.37+0.10	1.95	1.20
HM(7,4)	255§/7.3§	0.12§ = 0.03§+0.09§	255/7.3	0.08 = 0.01+0.07	1	1
HM(15,11)	4232§/13.8§	56.82§ = 55.86§+0.96§	3279/13.2	1.33 = 0.35+0.98	0.77	0.96

Table 2: Comparison on Decoder Existence Checking and Decoder Generation.

circuit	[15]				Decosy				area ratio	delay ratio
	$(n, p, n^\dagger, p^\dagger)$	#in/#reg	area/delay	time	$(n, p, n^\dagger, p^\dagger)$	#in/#reg	area/delay	time		
XGXS	$(1, 1, -1, 1)$	11/0	293/7.5	3.31 (2.70)	$(1, 1, -1, 1)$	11/0	295/7.1	0.07	1.01	0.95
XFI	$(3, 0, 1, 0)$	67/66	5697/14.4	1001.77 (1144.32)	$(3, 1, 1, 0)$	67/66	3913/12.5	8.59	0.69	0.87
Scrambler	$(2, 0, 1, 0)$	65/64	736/3.8	13.55 (10.46)	$(1, 1, 1, 0)$	65/64	640/3.8	0.42	0.87	1
PCIE	$(1, 2, -2, 2)$	11/0	163/6.1	5.1 (3.91)	$(1, 2, -2, 2)$	11/0	190/6.6	0.07	1.17	1.08
T2Ethernet	$(1, 4, -4, 4)$	11/0	269/6.9	137.26 (113.89)	$(1, 4, -4, 4)$	11/0	526/9.7	1.81	1.96	1.41
HM(7,4)	$(0, 0, 0, 0)$	7/0	255§/7.3§	0.12§ (NA)	$(0, 0, 0, 0)$	7/0	255/7.3	0.05	1	1
HM(15,11)	$(0, 0, 0, 0)$	15/0	4232§/13.8§	56.92§ (NA)	$(0, 0, 0, 0)$	15/0	3279/13.2	2.02	0.77	0.96

Table 3: Comparison on Decoder Existence Checking.

circuit (w/o decoder)	[15]	Decosy
	time	time
XGXS_err	2.17 (1.23)	0.01
XFI_err	39.71 (44.58)	0.01
Scrambler_err	3.96 (3.26)	0.08
PCIE_err	2.94 (1.67)	0.01
T2Ethernet_err	128.73 (21.49)	0.04
HM(7,4)_err	1.35 (NA)	0.01
HM(15,11)_err	3.01 (NA)	0.01
AD	> 6000 (NA)	0.01

to automatic decoder synthesis. Experiments showed that our method, based on incremental SAT-solving and Craig interpolation, effectively determined decoder (non)existence and generated decoders, if they exist. To optimize decoder, using a script of synthesis commands has turned out to be effective, despite potential further improvements. The synthesized decoders exhibit qualities comparable to prior work, which equipped with XOR-based decoder optimization. Hence our approach may potentially benefit the design and verification of encoding/decoding systems in various applications.

Acknowledgments

This work was supported in part by the National Science Council under grants NSC 99-2221-E-002-214-MY3 and NSC 99-2923-E-002-005-MY3.

7. REFERENCES

[1] D. Bertozzi, L. Benini, and G. De Micheli. Low power error resilient encoding for on-chip data buses. In *Proc. Design, Automation and Test in Europe (DATE)*, pages 102-109, 2002.

[2] A. Biere, A. Cimatti, E. Clarke, Y. Zhu. Symbolic model checking without BDDs. In *Proc. Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 193-207, 1999.

[3] Berkeley Logic Synthesis and Verification Group. *ABC: A system for sequential synthesis and verification.* `http://www.eecs.berkeley.edu/~alanmi/abc/`

[4] W. Craig. Three uses of the Herbrand-Gentzen theorem in relating model theory and proof theory. *J. Symbolic Logic*, 22(3):269-285, 1957.

[5] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 502-518, 2003.

[6] `http://www.ssypub.org/` (access date: September, 2010)

[7] J.-H. R. Jiang, H.-P. Lin, and W.-L. Hung. Interpolating functions from large Boolean relations. In *Proc. Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 779-784, 2009.

[8] J.-H. R. Jiang, C.-C. Lee, A. Mishchenko, and C.-Y. Huang. To SAT or not to SAT: Scalable exploration of functional dependency. *IEEE Trans. on Computers*, 59(4):457-467, April 2010.

[9] K. McMillan. Interpolation and SAT-based model checking. In *Proc. Int'l Conf. on Computer Aided Verification (CAV)*, pages 1-13, 2003.

[10] M. Moskewicz, C. Madigan, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. Design Automation Conference (DAC)*, pages 530-535, 2001.

[11] T. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*, Wiley-Interscience, 2005.

[12] K. Sayood. *Introduction to Data Compression*, 3rd edition, Morgan Kaufmann, 2005.

[13] S. Sridhara and N. Shanbhag. Coding for system-on-chip networks: A unified framework. *IEEE Trans. on VLSI Systems*, 13(6):655-667, 2005.

[14] S. Shen, Y. Qin, K. Wang, L. Xiao, J. Zhang, and S. Li. Synthesizing complementary circuits automatically. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 29(8):1191-1202, August 2010.

[15] S. Shen, Y. Qin, J. Zhang, and S. Li. A halting algorithm to determine the existence of decoder. In *Proc. Formal Methods in Computer Aided Design (FMCAD)*, pages 91-99, 2010.

[16] W. Trappe and L. Washington. *Introduction to Cryptography with Coding Theory*, 2nd edition, Prentice Hall, 2005.