Automatic Decoder Synthesis: Methods and Case Studies

Hsiou-Yuan Liu, Yen-Cheng Chou, Chen-Hsuan Lin, and Jie-Hong R. Jiang, *Member, IEEE*

***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 non-existence instantly while prior methods may fail. Case studies are also conducted in synthesizing decoders for linear error-correcting codes.**

I. 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 [15], to make messages unintelligible in cryptography [23], to compress information in data processing and storage [17], to be resilient to soft errors in chip design [1], [19], and so on. Despite the diversity, encoding/decoding systems can often be modelled as finite state machines, e.g., convolutional codes in error correction, line codes in Ethernet and RFID, stream ciphers in symmetric encryption, etc. (Note that memoryless, or combinational, encoding/decoding can be thought of as a single-state finite state machine.) 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.

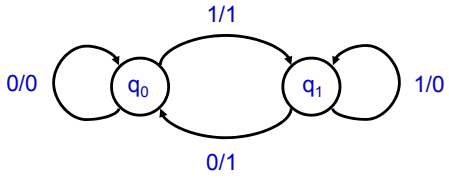

Fig. 1. A 0-1 alternation detector with unspecified initial states.

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. [20], [21] studied the decoder synthesis problem. A *bounded* decoder existence checking method was proposed [20], 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 [21] 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. The problem results from the misconception that the notion of *unique states* [21] 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. 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

Manuscript received September XX, 20XX; revised November XX, 20XX. A preliminary version of this paper appeared in [12].

H.-Y. Liu and Y.-C. Chou were with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan.

C.-H. Lin was with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan.

J.-H. R. Jiang is with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan (E-mail: jhjiang@cc.ee.ntu.edu.tw).

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 decoder derivable along the existence checking. (Shen et al. [20] suggested as future work using interpolation-based relation determinization [10] for decoder generation. Our interpolation formulation for decoder synthesis can be more direct and simpler than the prior method [10].) Fourthly, 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. Finally, we extend our method to synthesize error-correcting decoders of Hamming codes and convolutional codes. 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 II gives the preliminaries. Our main results on decoder existence checking and synthesis are presented in Section III. Implementation issues are discussed in Section IV. Extensions to synthesizing error-correcting decoders are studied in Section V. The proposed methods are evaluated with experimental results in Section VI. Section VII discusses our experience in decoder synthesis and verification, and potential decoder simplification. Finally, Section VIII concludes this paper and outlines future work.

II. 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., $[\![(x_1, x_2)]\!] = \{(0,0), (0,1), (1,0), (1,1)\}$.

A. SAT Solving and Craig Interpolation

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. Without loss of generality, we shall assume there are no repeated or complementary literals appearing in the same clause. A *SAT instance* is a conjunction of clauses, i.e., 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 SAT instance is *satisfiable* if there exists a satisfying assignment such that the CNF formula evaluates to 1. Otherwise it is *unsatisfiable*.

1) Refutation Proof and Craig Interpolation: Assume literal v is in clause C_1 and $\neg v$ in C_2. A *resolution* of clauses C_1 and C_2 on variable v yields a new clause C containing all literals in C_1 and C_2 except for v and $\neg v$. The clause C is called the *resolvent* of C_1 and C_2, and variable v the *pivot variable*.

For an unsatisfiable SAT instance, there always exists a sequence of resolution steps leading to an empty clause. Often only a subset of the clauses of a SAT instance participates in the resolution steps leading to an empty clause. This subset is the so-called *unsatisfiable core*.

A *refutation proof* Π of an unsatisfiable SAT instance S is a directed acyclic graph (DAG) $\Gamma = (N, A)$, where every node in N represents a clause which is either a root clause in S or a resolvent clause having exactly two predecessor nodes, and every arc in A connects a node to its ancestor node. The unique leaf of Π corresponds the empty clause. Modern SAT solvers, such as Chaff [14] and MiniSat [6], are capable of producing a refutation proof from an unsatisfiable SAT instance.

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 $\phi_A \Rightarrow \psi_A$ and $\psi_A \wedge \phi_B$ is 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 [13].

2) Circuit to CNF Conversion: Given a circuit netlist, it can be converted to a CNF formula in such a way that the satisfiability is preserved [22]. The conversion is achievable in linear time by introducing extra intermediate variables. In the sequel, we shall assume that the clause set of a Boolean formula ϕ is available from such conversion.

B. 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 state* 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.

C. 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 starting from 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 [20] 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.

III. Main Algorithms

A. Decoder Existence

1) Principles: 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 [20]): 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 $t = -n, \ldots, p$ for $n, p \geq 0$. Input $\vec{i}^0$ can be uniquely determined from $\vec{o}^{-n}, \ldots, \vec{o}^p$ 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 "$\neq$" asserts the corresponding 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.

Remark 1: For the sake of simplicity, Formula (1) assumes the (n, p) observation window covers the 0^{th} output $\vec{o}^0$. However it is not a necessity. In fact, a decoder observing only future outputs, i.e., $\vec{o}^t$ with $t > 0$, is possible to decide $\vec{i}^0$. On the other hand, there is no decoder that can refer only to past outputs, i.e., $\vec{o}^t$ with $t < 0$, since past encoder outputs have no influence on future (uncontrollable) encoder inputs, assuming the environment reacts independent of the encoder's response. Observation window minimization will be discussed in Section III-B.

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 $-n^{\text{th}}$ to p^{th} 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

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

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

Proof: ($\Leftarrow$) The encoder input cannot be uniquely determined by output sequences of bounded lengths as this infinite output sequence provides a counterexample.

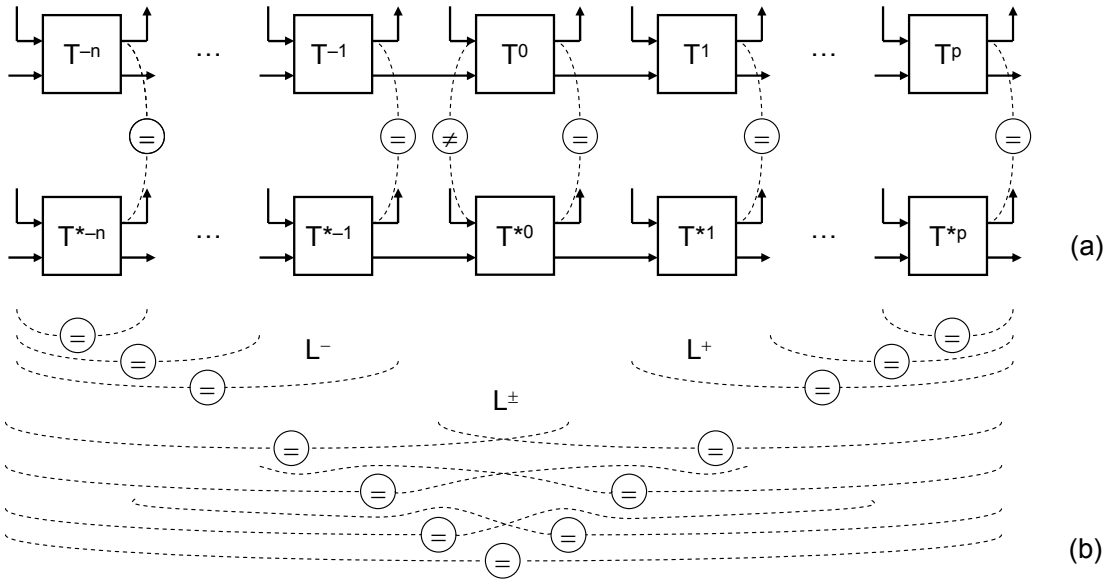

Fig. 2. (a) Decoder (n,p)-miter of transition system T; (b) looping constraints L^-, L^+, and $L^\pm$ for the state variables of the (n,p)-miter in (a).

($\Rightarrow$) 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 $t = l^* - 1, \ldots, u^* + 1$, 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

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

where

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

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

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

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 $[\![\vec{s}]\!] \times [\![\vec{s^*}]\!]$.

($\Leftarrow$) 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 $\vec{o}^{-n} = \vec{o^*}^{-n}, \ldots, \vec{o}^p = \vec{o^*}^p$ 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.

($\Rightarrow$) 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. ■

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.

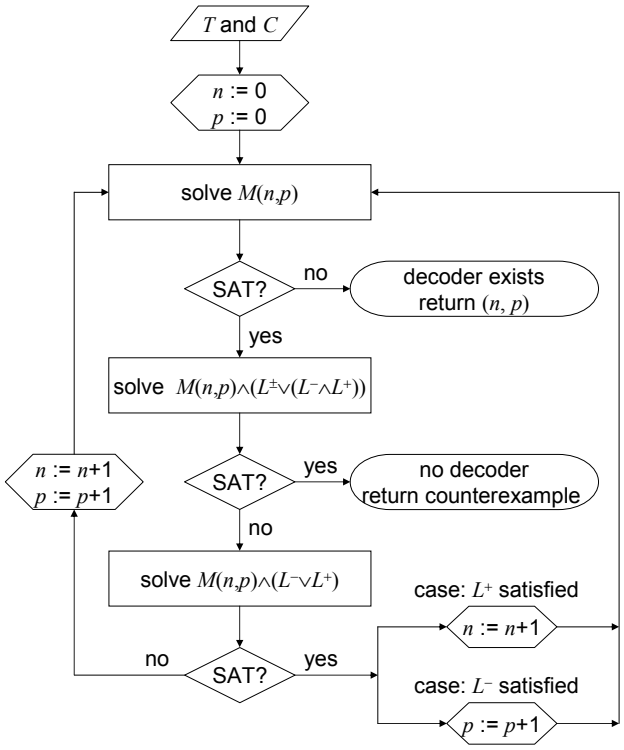

Fig. 3. Decoder existence checking.

2) Computation: 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. Nevertheless, for the decoder nonexistence case, the values (n,p) upon termination cannot largely deviate from minimal as the following proposition asserts.

Proposition 2: When decoders do not exist, the procedure of Figure 3 terminates with $n,p \leq (n^*+p^*+1)$ for any minimal looping condition (n^*,p^*), that is, Formula (2) is satisfiable under (n^*,p^*) but not under any (n',p') with $n' < n^*, p' \leq p^*$ or $n' \leq n^*, p' < p^*$.

Proof: Observe that, in the third SAT query of Figure 3, if $M(n,p) \wedge L^{-}$ (respectively $M(n,p) \wedge L^{+}$) is once satisfiable due to a permanent loop that eventually causes decoder nonexistence in the negative time-frames (respectively positive time-frames), the parameter n (respectively p) will be freezed and stop incrementing throughout the later iterations. Therefore if the parameter (n^*,p^*) corresponds to decoder nonexistence with two loops one in L^{-} and the other in L^{+}, then $n = n^*$ and $p = p^*$. On the other hand, if the parameter (n^*,p^*) corresponds to decoder nonexistence with one loop in $L^{\pm}$, then $n,p \leq (n^*+p^*+1)$ because the corresponding values of variables $(\vec{s}^{\,0}, \vec{s^*}^{\,0})$ must reoccur at $(\vec{s}^{\,-n-p-1}, \vec{s^*}^{\,-n-p-1})$ (so L^{-}_{n+p+1} is satisfied) and the corresponding values of variables $(\vec{s}^{\,1}, \vec{s^*}^{\,1})$ must reoccur at $(\vec{s}^{\,n+p+2}, \vec{s^*}^{\,n+p+2})$ (so L^{+}_{n+p+1} is satisfied). ■

Unfortunately, for the decoder existence case, it seems not easy to given a theoretical upper bound other than the trivial $n+p \leq |S_C|^2$.

On the other hand, the alternative procedure of Figure 4 yields a simple upper bound analysis.

Proposition 3: The procedure of Figure 4 terminates with $n,p \leq \max\{n^*,p^*\}$ for any minimal termination condition (n^*,p^*), that is, Formula (1) is unsatisfiable or Formula (2)

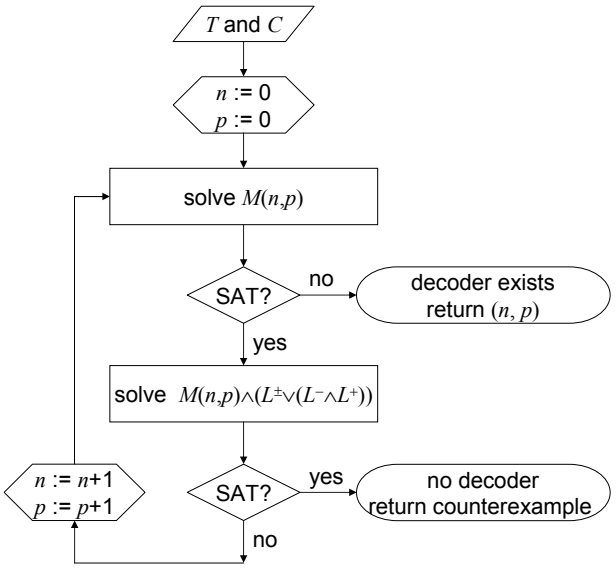

Fig. 4. Alternative decoder existence checking.

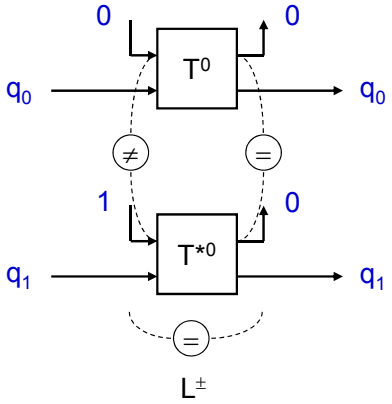

Fig. 5. Witness of decoder non-existence of the 0-1 alternation detector of Figure 1.

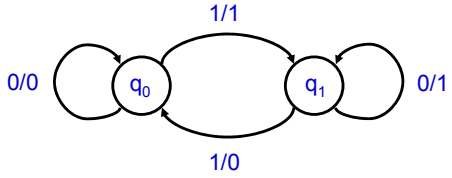

Fig. 6. A parity detector.

is satisfiable under (n^*, p^*) but not under any (n', p') with $n' < n^*, p' \leq p^*$ or $n' \leq n^*, p' < p^*$.

Proof: Since $n = p$ throughout the iterations, as long as n, p reach $\max\{n^*, p^*\}$ the procedure terminates. ■

Remark 2: The values of n and p do not necessarily correspond to the cost of decoder implementation. Essentially, not all the outputs of an observation window are necessary for decoding. There is room for decoder synthesis to explore a minimal subset of $\{\vec{o}^{-n}, \ldots, \vec{o}^{p}\}$ to decode $\vec{i}^{0}$.

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

$$L_{n,p}^{\pm} = \bigvee_{j=1}^{p+1} \left((\vec{s}^{-n} = \vec{s}^{j}) \wedge (\vec{s^*}^{-n} = \vec{s^*}^{j}) \right) \vee \bigvee_{j=-n}^{0} \left((\vec{s}^{p+1} = \vec{s}^{j}) \wedge (\vec{s^*}^{p+1} = \vec{s^*}^{j}) \right) \quad (6)$$

$$L_n^- = \bigvee_{j=-n+1}^{0} \left((\vec{s}^{-n} = \vec{s}^{j}) \wedge (\vec{s^*}^{-n} = \vec{s^*}^{j}) \right) \quad (7)$$

$$L_p^+ = \bigvee_{j=1}^{p} \left((\vec{s}^{p+1} = \vec{s}^{j}) \wedge (\vec{s^*}^{p+1} = \vec{s^*}^{j}) \right) \quad (8)$$

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 $M(n,p)$.

Example 1: Consider the 0-1 alternation detector of Figure 1. Formula $M(n,p) \wedge L_{n,p}^{\pm}$ with $n, p = 0$ is satisfiable by the truth assignments as shown in Figure 5. Thereby the algorithm of Figure 3 determines decoder non-existence without time-frame expansion.

Example 2: Consider the parity detector of Figure 6. It can be verified that formula $M(n,p)$ is satisfiable and $M(n,p) \wedge L_{n,p}^{\pm}$ is unsatisfiable for $n, p = 0$. (Recall that L_n^- and L_p^+ are defined to be false for $n, p = 0$.) In the subsequent iteration, $M(n,p)$ becomes unsatisfiable for $n, p = 1$. That is, the decoder exists. (In fact, $M(n,p)$ is readily unsatisfiable under $n = 1$ and $p = 0$.) To justify, for an arbitrary time index t we observe that output sequence $(o^{t-1}, o^t) = (0,0)$ uniquely determines input $i^t = 0$, $(0,1)$ determines $i^t = 1$, $(1,0)$ determines $i^t = 1$, and $(1,1)$ determines $i^t = 0$.

B. 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 $0 \leq p_i \leq p$ and $-p_i \leq n_i \leq n$, can be substantially reduced. Specifically, the formula

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

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 $\phi_{iA} \wedge \phi_{iB}$
04 **return** $(f_1, \ldots, f_{|\vec{x}|})$
end

Fig. 7. Algorithm: Decoder synthesis.

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.

Proposition 4: 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 $i = 1, \ldots, |\vec{x}|$.

Proof: Since $M(n, p) = \bigvee_{i=1}^{|\vec{x}|} M_i(n, p, n, p)$, where $i = 1, \ldots, |\vec{x}|$, the proposition follows. ■

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 [11] 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_{iA} and ϕ_{iB} be

$$\phi_{iA}: \quad \bigwedge_{t=-n}^{p} T^t \wedge \bigwedge_{t=-n}^{p+1} S_C^t \wedge x_i^0, \text{ and} \tag{10}$$

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

Then the interpolant ψ_{iA} of ϕ_{iA} with respect to ϕ_{iB} 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 $\phi_{iA} \wedge \phi_{iB}$ are equisatisfiable. So $\phi_{iA} \wedge \phi_{iB}$ is unsatisfiable.

By Theorem 1, we know that the interpolant ψ_{iA} refers only to $\vec{y}^{-n_i}, \ldots, \vec{y}^{p_i}$, the common variables of ϕ_{iA} and ϕ_{iB}. For $\phi_{iA} \Rightarrow \psi_{iA}$ 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_{iA} satisfiable and thus asserts x_i^0 will be in the onset of ψ_{iA}. For $\psi_{iA} \wedge \phi_{iB}$ 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_{iA}. On the other hand, since $\phi_{iA} \wedge \phi_{iB}$ is unsatisfiable, there is no output sequence $\vec{o}^{-n_i}, \ldots, \vec{o}^{p_i}$ in both onset and offset of ψ_{iA}. Hence ψ_{iA} 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 7.

Example 3: Consider the parity detector of Example2. Its transition relation $T(x, s, y, s')$ can be constructed as $(s' = x \oplus s) \wedge (y = x \oplus s)$, where symbol "$\oplus$" stands for the XOR operator. Let $\phi_A = T(x^{-1}, s^{-1}, y^{-1}, s^0) \wedge T(x^0, s^0, y^0, s^1) \wedge T(x^1, s^1, y^1, s^2) \wedge x^0$ and $\phi_B = T(x^{*-1}, s^{*-1}, y^{*-1}, s^{*0}) \wedge T(x^{*0}, s^{*0}, y^{*0}, s^{*1}) \wedge T(x^{*1}, s^{*1}, y^{*1}, s^{*2}) \wedge (y^{*-1} = y^{-1}) \wedge (y^{*0} = y^0) \wedge (y^{*1} = y^1) \wedge \neg x^{*0}$. Then interpolant $\psi_A = y^{-1} \oplus y^0$, derived from the refutation proof of $\phi_A \wedge \phi_B$, corresponds to the decoder.

IV. Implementation Details

We discuss two implementation issues and their solutions. As a result, the SAT solving procedures of decoder existence checking and synthesis are made very effective.

A. CNF Encoding of Disjunctive Constraints

Disjunctive constraints are usually not easily convertible to CNF. Unfortunately we encounter disjunctive constraints in two places: the vector inequality of $M(n, p)$, i.e., $\vec{x}^0 \neq \vec{x^*}^0$, equivalently $\bigvee_i (x_i^0 \neq x_i^{*0})$, and the looping constraints of Formula (2).

One popular way of encoding disjunctive constraints to CNF is to represent disjunction using OR-gates and then perform circuit-to-CNF conversion. The main difficulty arises from the requirement of representing the constraints to be disjuncted as circuits as well even if they are readily in CNF. Complication adds on to further support incremental SAT solving. Rather we propose a simple solution without relying on circuit conversion.

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 5: 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 6: Formula (13) and formula $\Phi_\ell \wedge b_\ell$ are equisatisfiable.

Note that, since literal b_ℓ can be asserted by *unit assumption* [6], Formula (14) is extendable to arbitrary ℓ for incremental solving. Thereby the looping constraints can be incrementally expressed in CNF.

Example 4: Consider solving two SAT instances $C_1 \wedge C_2 \wedge C_3$ and $(C_1 \wedge C_2 \wedge C_3) \vee (C_4 \wedge C_5)$ in a row. By the above construction, we have

$$\Phi_1 = (C_1 \vee \neg c_1) \wedge (C_2 \vee \neg c_1) \wedge (C_3 \vee \neg c_1) \wedge (c_1 \vee \neg b_1),$$
$$\Phi_2 = \Phi_1 \wedge (C_4 \vee \neg c_2) \wedge (C_5 \vee \neg c_2) \wedge (b_1 \vee c_2 \vee \neg b_2).$$

To solve the first SAT instance, letting b_1 be in the unit assumption of Φ_1 makes c_1 be implied. Thus Φ_1 reduces to the original formula $C_1 \wedge C_2 \wedge C_3$. Subsequently to solve the second SAT instance, we make unit assumption on b_2 in Φ_2, which in turn reduces to $\Phi_2' = (C_1 \vee \neg c_1) \wedge (C_2 \vee \neg c_1) \wedge (C_3 \vee \neg c_1) \wedge (c_1 \vee \neg b_1) \wedge (C_4 \vee \neg c_2) \wedge (C_5 \vee \neg c_2) \wedge (b_1 \vee c_2)$. The equisatisfiability between this formula and the second SAT instance can be seen from Table I.

B. Incremental Time-Frame Expansion

Incremental time-frame expansion is commonly applied in bounded and unbounded model checking. The incremental approach works for decoder existence checking as they bear similar problem structures. 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.

V. Case Studies

Our above discussion assumes the decoder receives accurate information from the encoder. We extend our horizon to study how our approach can be applied to synthesize error-correcting codes [15]. That is, the decoder may possibly receive noisy information, contaminated by an unreliable communication channel, from the encoder, and recovers the original message if the error is correctable. In particular, two families of *linear codes* are studied, including *Hamming codes* and *convolutional codes*.

A. Hamming Codes

Hamming codes are the first important error-correcting codes discovered in 1950 by Hamming [7]. They are widely used in computer random-access memory and other applications. Specifically the Hamming($2^m - 1$, $2^m - m - 1$) code, for $m \geq 2$, contains in total $2^m - 1$ bits, which consist of m parity bits and $2^m - m - 1$ data bits. It can correct up to any 1-bit error among the $2^m - 1$ bits.

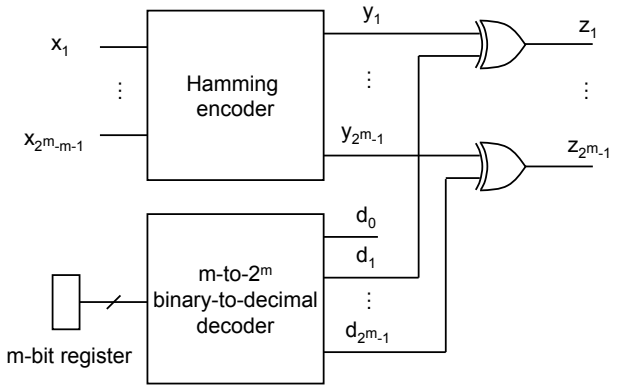

Fig. 8. Hamming encoder embedded with a noise source.

Essentially Hamming encoding and decoding are combinational, i.e., stateless. It is thus a special case of the decoder synthesis problem. However directly applying our synthesis algorithm on a Hamming encoder yields an inverse function without any error-correction capability. To synthesize an error-correcting Hamming decoder, the noise model must be embedded into the encoder.

To embed the noise model into a Hamming($2^m - 1$, $2^m - m - 1$) encoder, we introduce auxiliary registers and logic components to the encoder as shown in Figure 8. In the figure, m auxiliary registers are added, which feed to an m-to-2^m binary-to-decimal decoder. Essentially their values determine whether an error occurs and, if it occurs, which one of the code bits is flipped. (When the m register values are all 0, the circuit outputs are error-free. In other cases, exactly one of the circuit outputs is erroneous.) Since at most one of the binary-to-decimal decoder outputs valuates to logic value 1, at most one of the Hamming encoder outputs is flipped.

Since the variables correspond to the auxiliary registers are free (with values unspecified), the SAT solving of the decoder synthesis essentially explores all 2^m truth assignments. Therefore the synthesized decoder is capable of correcting up to any 1-bit errors as the following proposition asserts.

Proposition 7: The synthesized decoder of the noisy Hamming encoder of Figure 8 can correct up to any 1-bit error.

B. Convolutional Codes

Convolutional codes, introduced by Elias in 1955 [5], are another popular family of error-correcting codes. They differ from Hamming codes (generally speaking, block codes), where disjoint blocks of an input message are encoded, in that non-disjoint blocks of an input message are processed in a streaming manner.

A convolutional code is commonly specified by two parameters: its *code rate* r and *constraint length* ℓ. Let p be the number of input bits, q the number of output bits, and m the number of registers; then $r = p/q$ and $\ell = p(m-1)$. Except for deep-space applications, typical code rates range from $1/8$ to $7/8$ with p and q from 1 to 8, and m from 2 to 10. As

TABLE I
TRUTH TABLE FOR EQUISATISFIABILITY ANALYSIS.

$\bigwedge_{i=1}^{3} C_i$	$\bigwedge_{i=4}^{5} C_i$	$\bigwedge_{i=1}^{3} C_i \vee \bigwedge_{i=4}^{5} C_i$	satisfying assignments to (c_1, c_2, b_1) in Φ_2'
True	True	True	$\{010, 101, 110, 111\}$
True	False	True	$\{101\}$
False	True	True	$\{010\}$
False	False	False	$\{\}$

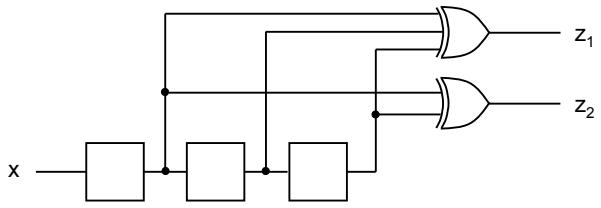

Fig. 9. A convolutional code encoder.

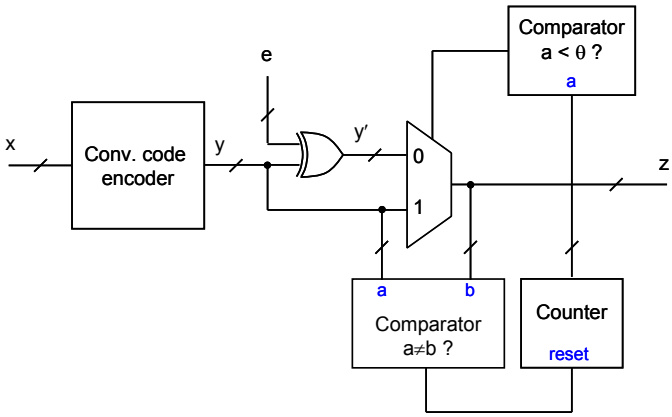

Fig. 10. Convolutional code encoder embedded with a noise source.

an example, Figure 9 shows a rate-$1/2$ convolutional code encoder with 3 registers.

Since the standard decoding of convolutional codes often involves maximum-likelihood decisions for probabilistic error recovery, SAT-based decoder synthesis cannot be directly applicable. Rather we cast the decoding problem in a different scenario considering a communication channel with sparse error injection. We pursue (conditional) exact data recovery, provided that any two errors do not occur within k consecutive data transmissions. When the errors are sparse, the synthesized decoder provides correct decoding with high probability. The assumption is valid for a bursty channel as well since interleavers can be applied on the encoder side to interleave burst errors [15].

To synthesize the decoder with error correcting capability, we embed an error source into the convolutional code encoder as shown in Figure 10. In the figure, the original encoder has inputs $\vec{x}$ and outputs y. The pseudo primary inputs $\vec{e}$, with $|\vec{e}| = |\vec{y}|$, are introduced to flip at most 1-bit of the values of $\vec{y}$. The original outputs $\vec{y}$ and the contaminated outputs $\vec{y}'$ are multiplexed by a to-be-specified control signal as the primary outputs of the entire circuit. To specified the multiplexer control signal, the lower comparator checks whether the multiplexer output differs from $\vec{y}$. If yes, it resets the counter value to zero, denoting that some error has taken effect. The upper comparator checks whether the counter value is less than some pre-defined constant θ. If yes, it enforces the multiplexer to select uncontaminated outputs $\vec{y}$ to ensure two adjacent errors cannot happen within θ time-frames. (In our experiment, $\theta = 4$.)

Note that, unlike the error source being introduced from the registers in the noisy Hamming encoder of Figure 8, the error of the noisy convolutional code encoder of Figure 10 is inserted by the pseudo inputs $\vec{e}$. Due to this difference, an error can possibly be introduced in every time-frame in the convolutional code encoder. In addition, the counter and comparators ensure that two adjacent errors are separated by at least $(\theta + 1)$ time-frames. Since any 1-bit error within a $(\theta + 1)$ time-frame window can possibly happen in the circuit of Figure 10, the synthesized decoder, if exists, can correct up to any 1-bit error within a $(\theta + 1)$ time-frame window.

Proposition 8: The synthesized decoder of the noisy convolutional code encoder of Figure 10 can correct up to any 1-bit error in a $(\theta + 1)$ time-frame window.

As mentioned earlier, although the assumption that no two adjacent errors occur within some k time-frames may not always hold for practical communication channels, as long as the errors are sparse our synthesized decoder can still be used for error recovery with high fidelity similar to the standard maximum likelihood decoding scheme.

VI. 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 [20], [21] were obtained online [8]. The benchmark circuits and their statistics (including gate counts, circuit levels, register counts, and datapath widths) are listed in Table II. The profiles of circuits `XGXS`, `XFI`, `Scrambler`, `PCIE`, and `T2Ethernet` can be found in [20]. Three additional designs: the `HM` series, implementing the Hamming code encoder of Section V, the `CC` series, implementing the (rate-$1/2$) convolutional code encoder of Section V, 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 [20] on decoder generation in Table III, comparison with [21] on decoder existence checking and generation in Table IV, and comparison with [21] on decoder existence checking for circuits without decoders in Table V. Note that the executables

TABLE II
BENCHMARK STATISTICS.

circuit	#gate/#level	#reg	datapath width
XGXS	177/15	15	8
XFI	2385/56	135	64
Scrambler	535/9	58	66
PCIE	265/24	22	10
T2Ethernet	1094/18	48	10
HM(7,4)	50/6	3	4
HM(15,11)	144/14	4	11
HM(31,26)	362/30	5	26
HM(63,57)	852/62	6	57
HM(127,120)	1934/126	7	120
HM(255,247)	4296/254	8	247
CC3	57/12	6	1
CC4	69/12	7	1
AD	5/2	1	1

of [20], [21] were implemented in OCaml running zChaff [14], whereas ours were implemented in C running MiniSat [6]. The reported runtimes in [20] and [21] were obtained on a CentOS 5.2 Linux machine with AMD Athlon 64 X2 dual core 2.4GHz CPU and 6GB RAM, and with Intel Core 2 Q6600 2.4GHz CPU and 8GB RAM, respectively. They were repeated in the parentheses in the fifth column of Table IV and the second column of Table V for reference. It is interesting to notice the curious runtime inconsistencies.

Table III compares decoder generation results of [20] and DECOSY with respect to the pre-specified parameters given in [20], 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 [20] 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 [20] were likely taking effect (as noted in [20] 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 III as well as in Table IV, were obtained by our own re-implementation of [20] for referential purposes.) As the decoders of `HM(31,26)` and above computed by any means were too large to be practically optimized by the ABC script, they were not listed.

To compare the synthesized decoders against their manually designed counterparts, we consider the Hamming decoders. The area/delay of the manually designed `HM(7,4)` (respectively `HM(15,11)`) decoder is $96.00/8.70$ (respectively $354.00/16.40$). In comparison, the synthesized Hamming decoders are slightly better in delay but much worse in area. (Comparisons for convolutional code decoders are not performed due to our non-standard interpretation of the decoding scheme.) For other circuits, similar comparisons can be found in [20]. Note that, even if a synthesized decoder can be sub-optimal, they can be still useful in the verification purpose.

Table IV compares the results of [21] 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 III-B. The runtime includes checking decoder existence and script optimization, same as those of Table III. The results are similar to those in Table III.

Table V compares the runtimes (in seconds) of [21] 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 [20]. 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 (by default with the aforementioned $L^{\pm}_{n,p}$ constraint enforced) 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 [21] is incomplete and less effective in detecting decoder non-existence. (The prior method [21] encountered a memory problem when executing `HM(255,247)_err`.) On the other hand, to study the usefulness of the (optional) $L^{\pm}_{n,p}$ constraint, it can be seen, by contrasting the last two columns of Table V, that not only the number of expanded time-frames but also the runtime can be effectively reduced.

To study the scalability of our method, the parametric `HM` and `HM_err` series circuits can be useful. Figure 11 plots the relationship between the datapath width and the runtime for decoder (non)existence checking of the Hamming coding scheme. Since the prior method (even with our re-implementation for the decoder-existent cases) is less effective, it was not included in this plot. These two curves suggested that existence checking (even without decoder generation) could be in general more sophisticated than nonexistence checking. Moreover, it is somewhat surprising to observe that the runtime of checking `HM(15,11)` is worse than that of `HM(31,26)`. This phenomenon might result from the unpredictability of SAT solver performance.

VII. DISCUSSIONS

We discuss some issues about our experience reimplementing the prior decoder generation algorithm [20], synthesizing decoders for error-correcting codes, verifying decoder correctness, and potential decoder minimization.

A. Prior Work Reimplementation

The decoder construction algorithm by Shen et al. [20] enumerated all satisfying assignments (ALLSAT) to, in our

TABLE III
COMPARISON ON DECODER GENERATION.

circuit	[20]		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 0.04+0.04	1.06	0.99
XFI	5697/14.4	492.58 490.45+2.13	3978/14.3	6.76 5.44+1.32	0.70	0.99
Scrambler	736/3.8	1.88 1.83+0.05	640/3.8	0.52 0.46+0.06	0.87	1.00
PCIE	171/5.8	1.04 1.02+0.02	190/6.6	0.12 0.10+0.02	1.11	1.14
T2Ethernet	299/7.5	22.67 22.62+0.05	583/9.0	2.09 1.92+0.17	1.95	1.20
HM(7,4)	$255^{\S}/7.3^{\S}$	$0.12^{\S}$ $0.03^{\S}$+$0.09^{\S}$	255/7.3	0.07 0.01+0.06	1.00	1.00
HM(15,11)	$4232^{\S}/13.8^{\S}$	$56.82^{\S}$ $55.86^{\S}$+$0.96^{\S}$	3279/13.2	2.38 0.52+1.86	0.77	0.96
CC3	$113^{\S}/8.2^{\S}$	$0.15^{\S}$ $0.03^{\S}$+$0.12^{\S}$	104/9.1	0.20 0.01+0.19	0.92	1.11
CC4	$125^{\S}/9.0^{\S}$	$0.17^{\S}$ $0.06^{\S}$+$0.11^{\S}$	129/9.0	0.18 0.01+0.17	1.03	1.00

TABLE IV
COMPARISON ON DECODER EXISTENCE CHECKING AND DECODER GENERATION.

circuit	[21]				DECOSY				area	delay
	$(n, p, n^{\dagger}, p^{\dagger})$	#in/#reg	area/delay	time	$(n, p, n^{\dagger}, p^{\dagger})$	#in/#reg	area/delay	time	ratio	ratio
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	1.00
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	$255^{\S}/7.3^{\S}$	$0.12^{\S}$ (NA)	$(0, 0, 0, 0)$	7/0	255/7.3	0.05	1.00	1.00
HM(15,11)	$(0, 0, 0, 0)$	15/0	$4232^{\S}/13.8^{\S}$	$56.92^{\S}$ (NA)	$(0, 0, 0, 0)$	15/0	3279/13.2	2.02	0.77	0.96
CC3	$(3, 2, 2, 2)$	10/0	$110^{\S}/7.0^{\S}$	$0.16^{\S}$ (NA)	$(2, 2, 2, 2)$	10/0	104/9.1	0.21	0.95	1.30
CC4	$(4, 3, 4, 3)$	16/0	$385^{\S}/11.6^{\S}$	$0.45^{\S}$ (NA)	$(1, 4, 1, 4)$	12/0	129/9.0	0.20	0.34	0.78

TABLE V
COMPARISON ON DECODER NONEXISTENCE CHECKING.

circuit (w/o decoder)	[21]	DECOSY	
	time	w/ $L^{\pm}$ time (n, p)	w/o $L^{\pm}$ time (n, p)
XGXS_err	2.17 (1.23)	0.01 (0, 0)	0.01 (1, 1)
XFI_err	39.71 (44.58)	0.01 (0, 0)	0.12 (1, 1)
Scrambler_err	3.96 (3.26)	0.08 (0, 0)	0.01 (1, 1)
PCIE_err	2.94 (1.67)	0.01 (0, 0)	0.01 (1, 1)
T2Ethernet_err	128.73 (21.49)	0.04 (0, 0)	0.01 (1, 1)
HM(7,4)_err	1.35 (NA)	0.01 (4, 4)	0.01 (8, 8)
HM(15,11)_err	3.01 (NA)	0.01 (4, 4)	0.09 (8, 8)
HM(31,26)_err	7.97 (NA)	0.04 (4, 4)	0.35 (8, 8)
HM(63,57)_err	23.25 (NA)	0.39 (4, 4)	1.04 (8, 8)
HM(127,120)_err	79.94 (NA)	4.13 (4, 4)	40.07 (8, 8)
HM(255,247)_err	– (NA)	478.74 (4, 4)	1646.61 (8, 8)
AD	> 6000 (NA)	0.01 (0, 0)	0.01 (1, 1)

notation, the formula

$$\bigwedge_{t=-n}^{p} T^t \wedge x_i^0, \text{ for } i = 1, \ldots, |\vec{x}^0|$$

with projection onto the output variables $\vec{y}^t$, for $t = -n, \ldots, p$. Let F be the formula $\bigwedge_{t=-n}^{p} T^t$. To simplify the ALLSAT process, the authors further proposed to simplify F by intersecting it with the unsatisfiable core of Formula (1). Let F' be the resultant simplified formula. Although this formula simplification can possibly improve decoder construction, it may occasionally yield completely negative effects as we discuss below.

Consider the decoder function of x_i^0. Its onset (respectively offset) corresponds to the assignments to the output variables $\vec{y}^t$, for $t = -n, \ldots, p$ that make formula $F \wedge x_i^0$ (respectively $F \wedge \neg x_i^0$) remain satisfiable. Its don't-care set, on the other

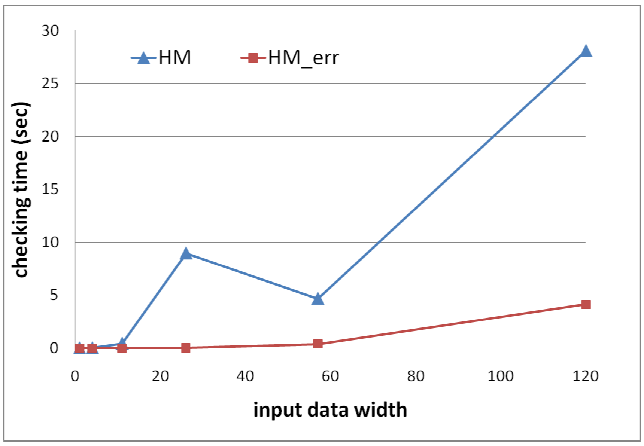

Fig. 11. Hamming data width vs. checking time.

hand, corresponds to those that make formula F unsatisfiable. By simplifying F to F', the effective onset and offset are increased whereas the don't-care set is reduced.

Such changes may cause two negative effects:

1) There are more satisfying assignments to enumerate because $F' \wedge x_i^0$ is easier to satisfy than $F \wedge x_i^0$.
2) Given a satisfying assignment to $F' \wedge x_i^0$, it is harder to minimize (either by removing the variable assignments irrelevant to the satisfiability, namely BFL_UNSAT in [20], or by searching XOR-gate implementation, namely XORMIN in [20]) because expanding the assignment becomes more likely to intersect with the increased offset.

These negative effects are much amplified especially when F' is substantially simplified from F. It is, therefore, a tradeoff to utilize what amount of the unsatisfiability core information although simplifying formula F may potentially reduce the efforts in individual SAT queries.

Based on our re-implementation of [20], we observed that the aforementioned effects are particularly noticeable in circuit `XFI`. Table VI shows the decoder generation times (without counting the time running the simplification script) under three different options of using unsatisfiable core information. The second column shows the runtime with F not being simplified, the third column shows that with F simplified as suggested in [20], and the fourth column shows that with F simplified by our modified unsatisfiable core reduction, which enlarges the unsatisfiable core by taking into account the unsatisfiable core clauses in T^{*t} of Formula (1). The results might reveal that the unsatisfiable core reduction of formula F can be sensitive to SAT solving and non-robust, although our re-implementation may not reflect how the reduction was actually implemented in [20].

TABLE VI
EFFECTS OF UNSATISFIABLE CORE REDUCTION ON CIRCUIT `XFI`.

	w/o UNSAT	w/ UNSAT	w/ Modified UNSAT
time	860.14	> 6000	51.33

On the other hand, in our proposed decoder generation method we observed, from empirical experience, no much gain from simplifying formula F with unsatisfiable core information.

B. Error-Correcting Decoder Synthesis

In our case studies on Hamming codes and convolutional codes, it became apparent that current SAT solvers are only effective in solving small instances, especially for the decoder existent cases. The abundance of XOR constraints makes SAT solvers inefficient. Neither our approach, nor prior approach is scalable. As an example, the decoder of `HM(31,26)` can be hardly generated due to inefficient SAT solving and large interpolants generated.

Although empirical experience suggests that encoder simplification may sometimes help reduce the efforts in decoder existence checking and synthesis, fundamental breakthroughs in SAT solving remain needed to cope with XOR constraints for practical synthesis of error-correcting decoders. Fortunately, there are recent attempts, such as CryptoMiniSat [18], that show possible solutions to the current limitation.

C. Correctness and Equivalence Verification

To verify the correctness of a synthesized decoder, we unroll the encoder with some number of required time-frames, and connect properly the expanded encoder outputs to the decoder inputs. The decoder is correct if the inputs of the expanded encoder at a specific time-frame are identical to the decoder outputs. That is, we check whether the composition of the encoder and decoder is equivalent to an identity function.

For a given encoder specification, there can be many different decoders generated. The notion of their equivalences differs from the conventional notions of combinational and even sequential equivalence. Therefore conventional equivalence checking does not apply. Rather they are equivalent in the sense that they reverse the behavior of the same encoder. Their equivalences can surely be established by verifying individual correctness as mentioned above. On the other hand, equivalence verification of decoders without given an encoder can be challenging.

D. Decoder Simplification

Due to the special problem structure, the synthesized decoders are in the form of pipelined circuits with shift registers. Because of the shift registers, the set of reachable states of a decoder circuit can be far smaller than the entire state set. Unlike general sequential circuits with feedbacks, state reachability of pipelined circuits can be analyzed efficiently. Therefore, sequential don't cares with unreachable state information can be exploited for decoder simplification. Moreover, it may be possible to simplify a decoder with equivalent state identification [9].

VIII. CONCLUSIONS AND FUTURE WORK

We have presented the first sound and complete approach 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. As a result, our approach may potentially benefit the design and verification of encoding/decoding systems in various applications.

As our current optimization did not exploit the fact that decoders are usually XOR-gate dominated designs, there is room for future improvement. As decoder synthesis is a new field, the available set of benchmark circuits is limited and remains to be expanded, especially, for the inclusion of large designs to probe the limitation and scalability of decoder synthesis algorithms. We also plan to perform more case studies on different encoding/decoding schemes.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Council under grants NSC 99-2221-E-002-214-MY3, 99-2923-E-002-005-MY3, and 100-2923-E-002-008.

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] P. Elias. Coding for noisy channels. *IRE Convention Record*, 3(4):37-46, 1955.

[6] 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.

[7] R. Hamming. Error detecting and error correcting codes. *Bell Syst. Tech. Journal*, vol. 29, pages 41-56, 1950.

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

[9] J.-H. R. Jiang and R. K. Brayton. On the verification of sequential equivalence. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):686-697, 2003.

[10] 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.

[11] 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.

[12] H.-Y. Liu, Y.-C. Chou, C.-H. Lin, and J.-H. R. Jiang. Towards completely automatic decoder synthesis. In *Proc. Int'l Conf. on Computer-Aided Design (ICCAD)*, pages 389-395, 2011.

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

[14] 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.

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

[16] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Computers*, 48(5):506-521, May 1999.

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

[18] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *Proc. Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT)*, pages 244-257, 2009.

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

[20] 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.

[21] 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.

[22] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, pages 115-125, 1970.

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