Design Verification of Sequential Machines Based on a Model Checking Algorithm of ε-free Regular Temporal Logic

Hiromi Hiraishi

September, 1988

CMU-CS-88-195

Hiromi Hiraishi
Dept. of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

This research was sponsored in part by the National Science Foundation under Grant Number CCR-8722633. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or of the U.S. Government.

Design Verification of Sequential Machines Based on a Model Checking Algorithm of ϵ-free Regular Temporal Logic

Hiromi Hiraishi

Abstract

In this paper an approach to design verification of sequential machine based on a model checking method of ϵ-free Regular Temporal Logic (ϵ-free RTL) is shown and its implementation and some verification examples are described. ϵ-free RTL is a linear time temporal logic whose expressive power is equivalent to ϵ-free regular set. Although the complexity of model checking problem of ϵ-free RTL is shown to be non-elementary, an efficient model checking algorithm which still runs in linear order of the size of structure models is proposed. ϵ-free RTL model checker based on the proposed algorithm shows that it is able to determine whether designed sequential machines of medium size of states satisfy their specifications in reasonable time and space.

The author is currently on leave from Department of Information Science, Kyoto University, Kyoto 606, JAPAN.

Design Verification of Sequential Machines Based on a Model Checking Algorithm of ϵ-free Regular Temporal Logic

Hiromi HIRAISHI*
Carnegie Mellon University, Pittsburgh

1. Introduction

According to the progress of VLSI technologies, it becomes important to establish new logic design methodologies which make us possible to verify correctness of logic design. Logic simulations do not meet this requirement because it cannot guarantee correctness of design in general. As one of the formal approaches to this goal, temporal logic[10] is now widely studied in many research field such as concurrent process and hardware design verification because it can be used for reasoning about event sequences.

Some practical design verification systems have been developed using temporal logic. Uehara et al. developed a DDL verifier[11] and Fujita et al. implemented a verification system in Prolog[6] by using traditional temporal logic. Clarke and Emerson proposed a new temporal logic called computation tree logic (CTL)[4], which combines both linear time and branching time temporal logic, and developed a CTL model checker[5] which runs linear in both size of specification and Kripke model.

We cannot characterize, however, a finite state machine completely by using these temporal logic because of their lack of expressive power; they cannot express regular set which is equivalent to finite state machine. In order to extend expressive power of temporal logic, Wolper et al. introduced temporal operators associated with right linear grammar and/or Büchi automata [13,14]. In order to describe specifications in their logic, however, it is necessary to *design* automata which satisfies specification in a sense. Thus it is easy to make same error both in specifications and its design. Furthermore, their approach to design verification is based on inclusion problems of two automata by extracting automata from specifications[12], and it is not easy to find out cause of design error from the verification result. Moszkowski proposed more powerful temporal logic named *interval temporal logic* (ITL)[9], but satisfiability problem of ITL is undecidable. Therefore, it is difficult to use ITL as a basis of logic design verification.

Considering above stated problems, *regular temporal logic (RTL)* which is expressively equivalent to regular set has been proposed by Hiraishi et al. [7]. The logic used here as a basis of design verification of sequential machines is a sub-class of RTL and is named *ϵ-free regular temporal logic*

*On leave from Department of Information Science, Kyoto University, Kyoto 606, Japan.

(ϵ-free RTL). It is a linear time temporal logic and its expressive power is shown to be equivalent to ϵ-free regular set. Unlike the traditional temporal logic, however, its semantics is defined along finite sequences of states. If we assume that specifications are given in terms of relations between input and output signals of a machine to be designed, specifications for any finite state machine can be written in ϵ-free RTL because possible input-output sequences of a finite machine can be characterized as a regular set.

As for a method of design verification, model checking approach, which has been also used in [5], is adopted in this paper. Although the model checking problem of ϵ-free RTL is shown to be non-elementary, an efficient model checking algorithm of ϵ-free RTL which is still linear in the size of *structure model* is proposed. The ϵ-free RTL model checking algorithm treats ϵ-free RTL formulas of specifications directly without converting them into automata. It successively generates ϵ-free RTL formulas which should be hold at next time on a given state machine something like a tabular method[10,13] used for satisfiability problem of temporal logic. If it detects some design error, it is more easy to reason about the cause of errors from ϵ-free RTL formula which fails to hold. Furthermore, an ϵ-free RTL model checker based on the proposed algorithm has been implemented and several sequential machines with medium size of states have been verified by the ϵ-free RTL model checker in reasonable time.

This paper is organized as follows: Section 2. introduces syntax and semantics of ϵ-free RTL and shows its expressive power. Section 3. discusses verification methods and introduces structure models and defines truth value of ϵ-free RTL on structure models. In Section 4. derivative of ϵ-free RTL formula is defined and its characteristics are shown. The derivation is the basic operation in model checking algorithm and it generates ϵ-free RTL formula which should hold at next time on a structure model. Section 5. describes a model checking algorithm and its order is shown to be linear in the size of structure model. Section 6. discusses the complexity of ϵ-free RTL model checking problem and its complexity is proved to be non-elementary. Section 7. explains the characteristics of the ϵ-free RTL model checker. Examples of design verification are also given and they show that the ϵ-free RTL model checker is useful from practical point of view. Section 8. concludes this paper with summarizing the results and giving future problems.

2. ϵ-free Regular Temporal Logic

2.1. Syntax and Semantics

ϵ-free regular temporal logic (ϵ-free RTL) is a linear time temporal logic. It contains 3 temporal operators: '$\bigcirc$', ':', and '$[:]$'. These operators intuitively represent 'next time', 'concatenation', and 'repeat' respectively.

Let AP be a set of atomic propositions. ϵ-free RTL formulas are defined inductively as follows:

- If $p \in AP$, then p is an ϵ-free RTL formula.
- If η is an ϵ-free RTL formula, then so are $(\neg\eta), (\bigcirc\eta)$, and $([:]\eta)$.
- If η and ξ are ϵ-free RTL formulas, then so are $(\eta \vee \xi)$ and $(\eta : \xi)$.

Semantics of ϵ-free RTL is defined based on a linear time model. However, it manipulates only

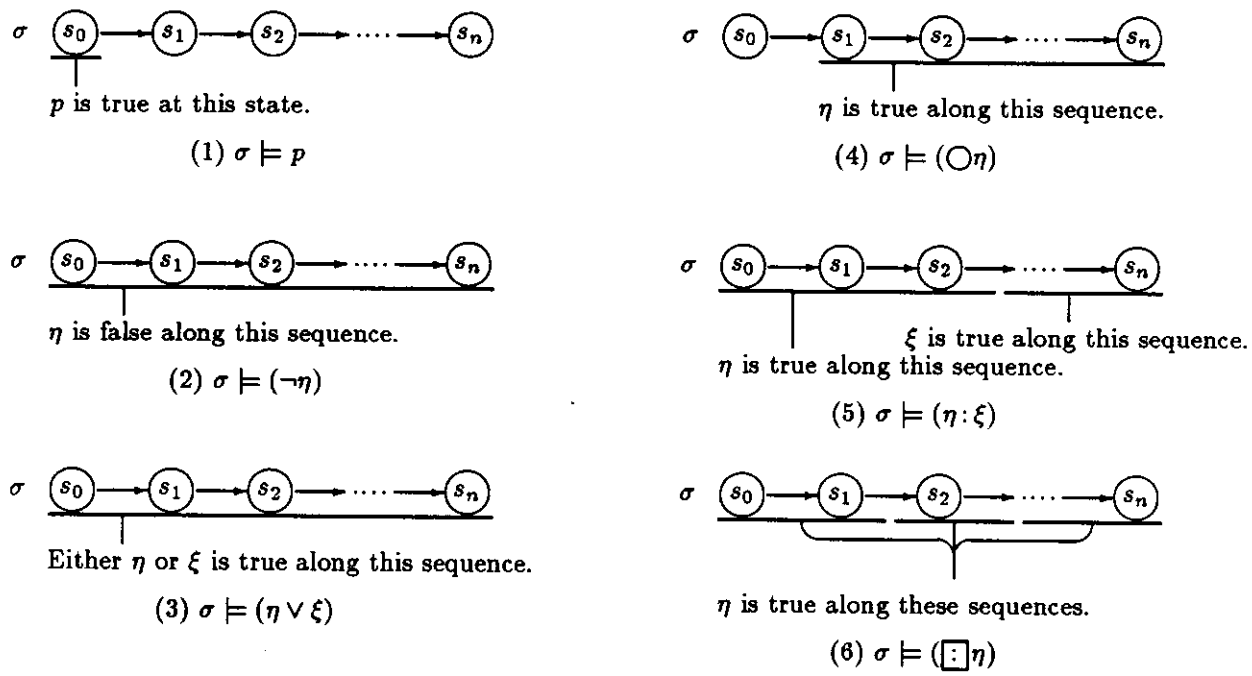


Figure 1: Intuitive Meaning of Logical Connectives

finite sequence of states while traditional linear time temporal logic manipulates infinite sequence of states. We define the semantics of ϵ-free RTL with respect to a linear model $M = (\Sigma, I)$, where

- Σ is a finite set of states.
- $I : \Sigma \rightarrow 2^{AP}$ is an interpretation function that labels each state with a set of atomic propositions true in that state.

Let $\sigma = s_0 s_1 \ldots s_n \in \Sigma^\dagger$ be a finite sequence of states. $\sigma(i)$ will denote the ith state in the sequence σ (i.e. $\sigma(i) = s_i$). $|\sigma|$ will denote the length of the sequence σ (i.e. $|\sigma| = n + 1$). In the case that $|\sigma| > i$, σ^i will denote the suffix sub-sequence of σ starting at s_i (i.e. $\sigma^i = s_i \ldots s_n$ and $\sigma^0 = \sigma$).

$M, \sigma \models \eta$ denotes that a formula η holds along sequence σ in linear model M. In the following we sometimes omit M and just write as $\sigma \models \eta$ if there is no confusion. Let p be an atomic proposition and η and ξ be formulas. The relation $\models$ is defined inductively as follows:

1. $\sigma \models p$ *iff* $p \in I(\sigma(0))$.
2. $\sigma \models (\neg\eta)$ *iff* $\sigma \not\models \eta$.
3. $\sigma \models (\eta \vee \xi)$ *iff* $\sigma \models \eta$ or $\sigma \models \xi$.
4. $\sigma \models (\bigcirc\eta)$ *iff* $|\eta| \geq 2$ and $\sigma^1 \models \eta$.
5. $\sigma \models (\eta : \xi)$ *iff* there exist $\sigma_1, \sigma_2 \in \Sigma^\dagger$ such that $\sigma = \sigma_1\sigma_2$, $\sigma_1 \models \eta$, and $\sigma_2 \models \xi$.
6. $\sigma \models ([:]\eta)$ *iff* there exists $\sigma_i \in \Sigma^\dagger$ such that $\sigma = \sigma_1\sigma_2 \ldots \sigma_m$ and $\sigma_i \models \eta$ $(1 \leq i \leq m \leq |\eta|)$.

An ϵ-free RTL formula η is said to be *satisfiable* iff (if and only if) there exist some linear model $M = (\Sigma, I)$ and some sequence $\sigma \in \Sigma^\dagger$ such that $M, \sigma \models \eta$.

Intuitive meaning of each logical connective for a sequence of states $\sigma = s_0 s_1 \ldots s_n$ are shown in Figure 1. '$\neg$' and '$\vee$' represent Boolean *negation* and *disjunction* respectively. $\bigcirc\eta$ means that η holds along the sequence starting from the next state. $\eta : \xi$ means that η holds along the first half of the sequence and ξ holds along the latter half of the sequence. $[:]\eta$ means that η holds repeatedly.

We will also use the following abbreviations in writing ϵ-free RTL formulas.

- $\eta \wedge \xi \stackrel{\text{def}}{=} (\neg((\neg\eta) \vee (\neg\xi)))$
- $\eta \Rightarrow \xi \stackrel{\text{def}}{=} ((\neg\eta) \vee \xi)$
- $\eta \equiv \xi \stackrel{\text{def}}{=} ((\eta \Rightarrow \xi) \wedge (\xi \Rightarrow \eta))$
- $\eta \oplus \xi \stackrel{\text{def}}{=} (\neg(\eta \equiv \xi))$
- $V_T \stackrel{\text{def}}{=} (\eta \vee (\neg\eta))$
- $V_F \stackrel{\text{def}}{=} (\neg V_T)$

'$\wedge$', '$\Rightarrow$', '$\equiv$', and '$\oplus$' represent Boolean *conjunction*, *implication*, *equivalence*, and *exclusive-or* respectively. 'V_T' and 'V_F' represent *tautology* and *invalid* formula respectively. Unary operators '$\neg$', '$\bigcirc$', and '$[:]$' have higher precedence than binary operators '$\vee$', '$\wedge$', '$\Rightarrow$', '$\equiv$','$\oplus$' and '$:$'. When there is no ambiguity, we usually omit parenthesis '(' and ')'.

2.2. Expressive Power

ϵ-free RTL can express various properties of sequences. For example, a set of sequences whose length are exactly 1 can be expressed by

$$LEN1 \stackrel{\text{def}}{=} \neg \bigcirc V_T = \neg(V_T : V_T) .$$

$\Diamond\eta$ which denotes that η holds at some point in a sequence and $\Box\eta$ which denotes that η holds at every point in a sequence can be defined as follow:

- $\Diamond\eta \stackrel{\text{def}}{=} \eta \vee (V_T : \eta)$
- $\Box\eta \stackrel{\text{def}}{=} \neg\Diamond\neg\eta = \eta \wedge \neg(V_T : \neg\eta)$.

The property that p is true at the first state and it is also true thereafter at every other state in a sequence, which cannot be expressed by traditional linear temporal logic [13], can be expressed as follows:

$$(p \wedge LEN1) \vee [:](p \wedge \bigcirc LEN1) \vee ([:](p \wedge \bigcirc LEN1) : LEN1) .$$

Furthermore, although we use 3 temporal operators in the definition of ϵ-free RTL, the temporal operator '$\bigcirc$' is redundant. $\bigcirc\eta$ can be expressed as

$$\bigcirc\eta = LEN1 : \eta = \neg(V_T : V_T) : \eta .$$

Next, we discuss relationship between an ϵ-free regular set and a set of sequences of states which can be expressed by a ϵ-free RTL formula, where an ϵ-free regular set is a regular set which does not contain null string ϵ. Let $L_{\Sigma,I}(\eta)$ be a set of sequences over Σ along which η holds with respect to the linear model (Σ, I). More precisely, $L_{\Sigma,I}(\eta) = \{\sigma | \sigma \in \Sigma^\dagger, \sigma \models \eta\}$. If there is no confusion, we abbreviate $L_{\Sigma,I}(\eta)$ as $L(\eta)$. $L_{\Sigma,I}(\eta)$ can be obtained as follows:

$$\eta \stackrel{\text{def}}{=} p_x \equiv (p_z \equiv \bigcirc \neg p_z)$$
$$spec \stackrel{\text{def}}{=} \neg p_z \wedge \Box(\eta \vee LEN1)$$

Sample I/O Sequence x: 0010111001
z: 0001101000

Figure 2: Specification of T Flipflop

- $L_{\Sigma,I}(p) = \{s | s \in \Sigma, p \in I(s)\}\Sigma^*$
- $L_{\Sigma,I}(\eta \vee \xi) = L_{\Sigma,I}(\eta) + L_{\Sigma,I}(\xi)$
- $L_{\Sigma,I}(\eta : \xi) = L_{\Sigma,I}(\eta) L_{\Sigma,I}(\xi)$
- $L_{\Sigma,I}(\neg\eta) = \Sigma^\dagger - L_{\Sigma,I}(\eta)$
- $L_{\Sigma,I}(\bigcirc\eta) = \Sigma L_{\Sigma,I}(\eta)$
- $L_{\Sigma,I}([\!:]\eta) = L_{\Sigma,I}(\eta)^\dagger$

On the other hand, for any ϵ-free regular set R over Σ, the corresponding ϵ-free RTL formula $F(R)$ such that $L_{\Sigma,I}(F(R)) = R$ can be constructed inductively by introducing corresponding atomic proposition p_s for each state $s \in \Sigma$ ($I(s) = \{p_s\}$) as follows:

- $F(\phi) = V_F$
- $F(R_1 + R_2) = F(R_1) \vee F(R_2)$
- $F(R^\dagger) = [\!:]F(R)$
- $F(s) = p_s \wedge LEN1$
- $F(R_1 R_2) = F(R_1) : F(R_2)$

Note that any ϵ-free regular set over Σ can be generated by applications of *union*('+'), *concatenation*, and *dagger*('†') operators over the alphabet Σ and an empty set ϕ. Therefore, we obtain the following theorem.

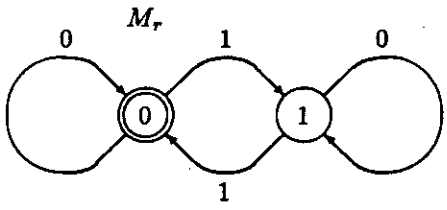
Theorem 1 *The expressive power of ϵ-free RTL is equivalent to ϵ-free regular set.*

3. Design Verification of Sequential Machines

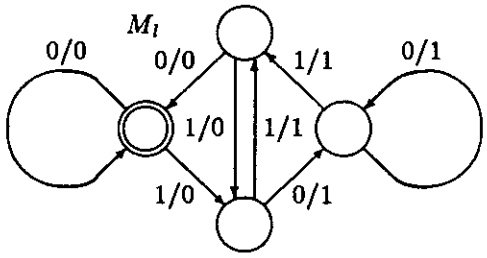
In this section we discuss how to verify design of sequential machines. Let us consider verification of a deterministic sequential machine with n binary input signals $X = \{x_1, x_2, \ldots, x_n\}$ and m binary output signals $Z = \{z_1, z_2, \ldots, z_m\}$.

We assume its specification is given in terms of possible input-output sequences of a machine to be designed. More precisely, a possible input-output sequence is a finite sequence ρ over $2^{X \cup Z}$ such that $x_i \in \rho(k)$ iff $x_i = 1$ at the kth input and $z_j \in \rho(k)$ iff $z_j = 1$ at the kth output ($1 \leq i \leq n, 1 \leq j \leq m$, and $0 \leq k < |\rho|$). Because the machine to be designed should have only finite number of states, the set of possible input-output sequences can be characterized as an ϵ-free regular set.

Let p_{x_i} and p_{z_j} be atomic propositions associated with input signal x_i and output signal z_j respectively such that p_{x_i} is *true* iff $x_i = 1$ and p_{z_j} is *true* iff $z_j = 1$. Then it is guaranteed by



Moore Type T Flipflop



Mealy Type T Flipflop

Figure 3: Two Designs of T Flipflop

Theorem 1 that there always exist an ϵ-free RTL formula *spec* which express the specification (i.e. the set of possible input-output sequences).

Figure 2 shows a specification *spec* of T flipflop written in ϵ-free RTL. We assume that the output z is 0 at initial. The output z changes its value iff the previous input x is 1. Since this property of T flipflop is meaningless for a time sequence whose length is 1, $LEN1$ is used in *spec* so that *spec* ignores non-existing next state at the last time of a time period under consideration.

We also assume that the result of design is given in the form of either a Moore machine or a Mealy machine (see Figure 3). Then the verification problem becomes to check if *spec* holds for all possible input-output sequences of the designed machine.

In order to treat possible input-output sequences more easily, we define a structure model of ϵ-free RTL. $K = (\Sigma, I, R, \Sigma_0)$ is called a *structure model* of ϵ-free RTL, where

- (Σ, I) is a linear model of ϵ-free RTL.
- $R \subseteq \Sigma \times \Sigma$ is a binary relation on Σ and denotes the possible transitions between states.
- $\Sigma_0 \subseteq \Sigma$ is a set of initial states.

A structure model is a kind of Kripke model[8] with a set of initial states where R is not necessarily a total relation. For a structure model $K = (\Sigma, I, R, \Sigma_0)$, a finite sequence of states $\pi = s_0 s_1 \ldots s_n$ is called a finite path from s_0 iff $(s_i, s_{i+1}) \in R$ for any i such that $0 \leq i < n$. Similarly, an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$ is called an infinite path from s_0 iff $(s_i, s_{i+1}) \in R$ for any $i \geq 0$.

Truth value of ϵ-free RTL formula with respect to a structure model K is defined as follows. An ϵ-free RTL formula η is said to be (K, s)-*true* if there exists a finite path π from s in the structure model K such that η holds along π ($\pi \models \eta$); it is said to be (K, s)-*false* otherwise. Furthermore, an ϵ-free RTL formula η is said to be K-*true* if there exists some initial state $s_0 \in \Sigma_0$ such that η is (K, s_0)-true; it is said to be K-*false* otherwise.

Let $M = (X, Z, S, \delta, \lambda, s_0)$ be a deterministic sequential machine with an initial state, where X, Z, and S are finite, nonempty sets of binary input signals, binary output signals, and states, respectively;

- s_0 is the initial state;
- $\delta : 2^X \times S \rightarrow S$ is the state transition function;
- λ is the output function such that
  - $\lambda : S \rightarrow 2^Z$ for Moore machine (We assume that λ is a total function.);
  - $\lambda : 2^X \times S \rightarrow 2^Z$ for Mealy machine (We assume that the λ is defined so long as δ is defined.).

A structure model K corresponding to a machine M is defined to be a structure model such that there exists one to one correspondence between the possible input-output sequences of M and the paths from one of an initial state in Σ_0 of K. Then design verification problem becomes to check if *spec* holds along any finite path from one of the initial states of the corresponding structure model K. In other words, $\neg spec$ is K-false iff the design is correct.

For a Moore machine $M_r = (X, Z, S, \delta, \lambda, s_0)$, its corresponding structure model $K_r = (\Sigma, I, R, \Sigma_0)$ is constructed as follows:

- $\Sigma = \{s'_{i,j} | s_i \in S, j \in 2^X$, and $\delta(j, s_i)$ is defined. $\}$
- $I(s'_{i,j}) = \{p_x | x \in j\} \cup \{p_z | z \in \lambda(s_i)\}$
- $R = \{(s_{i,j}, s_{i',j'}) | s_{i,j}, s_{i',j'} \in \Sigma, \delta(j, s_i) = s_{i'}\}$
- $\Sigma_0 = \{s'_{0,j} \in \Sigma\}$

The order of the number of states and transitions in K becomes as follows:

$$O(|\Sigma|) = O(|S|2^X),\ O(|R|) = O(|E|2^X),\text{ and } O(|\Sigma| + |R|) = O((|S| + |E|)2^X) = O(|S|2^{2|X|}).$$

For a Mealy machine $M_l = (X, Z, S, \delta, \lambda, s_0)$, its corresponding structure $K_l = (\Sigma, I, R, \Sigma_0)$ is constructed as follows:

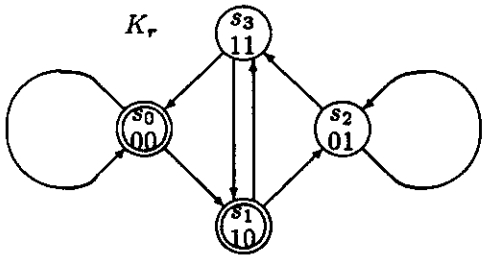
- $\Sigma = \{s'_{i,j,k} | s_i \in S, j \in 2^X, k \in 2^Z, \lambda(j, i) = k\}$
- $I(s'_{i,j,k}) = \{p_x | x \in j\} \cup \{p_z | z \in k\}$
- $R = \{(s_{i,j,k}, s_{i',j',k'}) | s_{i,j,k}, s_{i',j',k'} \in \Sigma, \delta(j, s_i) = s_{i'}\}$
- $\Sigma_0 = \{s'_{0,j,k} \in \Sigma\}$

The order of the number of states and transitions in K becomes as follows:

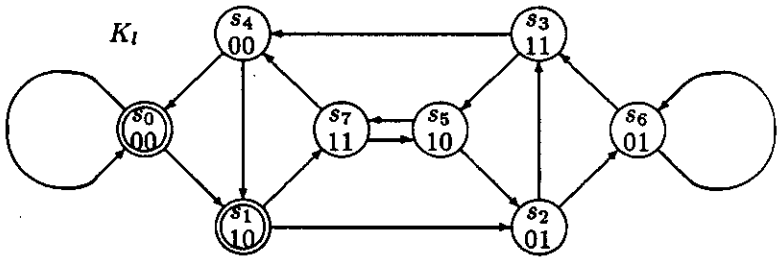
$$O(|\Sigma|) = O(|E|),\ O(|R|) = O(|E|2^X),\text{ and } O(|\Sigma| + |R|) = O(|E|(1 + 2^X)) = O(|S|2^{2|X|}).$$

If at least one next state is defined for each state in M_r and M_l, R of their corresponding structure model becomes a total relation.

Figure 4 shows the structure models corresponding to two designs (M_r and M_l; see Figure 3) of the T flipflop whose specification is given in Figure 2. K_r corresponds to M_r while K_l corresponds to M_r. Initial states are shown as double circles. Labels associated with nodes of structure models represent truth value of atomic propositions at the states in the order of p_x and p_z. K_l has equivalent states and can be reduced to K_r.



Structure Model Corresponding to M_r



Structure Model Corresponding to M_l

Figure 4: Structure Models of T Flipflop

4. Derivative of ϵ-free RTL Formulas

Derivative of an ϵ-free RTL formula η by a state $s \in \Sigma$, denoted by η/s, is defined inductively as follows:

$$
\begin{array}{lll}
1. & p/s & \stackrel{\text{def}}{=} \begin{cases} V_T & \text{if } p \in I(s) \\ V_F & \text{otherwise} \end{cases} \\
2. & (\neg\eta)/s & \stackrel{\text{def}}{=} \neg(\eta/s) \\
3. & (\eta \vee \xi)/s & \stackrel{\text{def}}{=} (\eta/s) \vee (\xi/s) \\
4. & (\bigcirc\eta)/s & \stackrel{\text{def}}{=} \eta \\
5. & (\eta : \xi)/s & \stackrel{\text{def}}{=} \begin{cases} \xi \vee ((\eta/s) : \xi) & \text{if } s \models \eta \\ (\eta/s) : \xi & \text{otherwise} \end{cases} \\
6. & (\boxed{:}\eta)/s & \stackrel{\text{def}}{=} \begin{cases} (\eta/s) \vee ((\eta/s) : \boxed{:}\eta) \vee \boxed{:}\eta & \text{if } s \models \eta \\ (\eta/s) \vee ((\eta/s) : \boxed{:}\eta) & \text{otherwise} \end{cases}
\end{array}
$$

By extending the above definition, a derivative by a sequence of states is defined as follows. Let η and $\sigma = s_0 s_1 \ldots s_n \in \Sigma^\dagger$ be an ϵ-free RTL formula and a sequence of states respectively. Then $\eta/\sigma \stackrel{\text{def}}{=} ((\cdots((\eta/s_0)/s_1)\cdots)/s_n)$. We also define that $\eta/\epsilon \stackrel{\text{def}}{=} \eta$, where ϵ is a null sequence.

As for derivatives, following equations hold, where $\odot$ denotes any Boolean operator with two arguments (i.e. $\wedge, \Rightarrow, \equiv, \oplus$, and so on):

- $(\eta \odot \xi)/s = (\eta/s) \odot (\xi/s)$
- $V_T/s = V_T$
- $V_F/s = V_F$
- $LEN1/s = V_F$
- $(\Diamond\eta)/s = (\eta/s) \vee \Diamond\eta$
- $(\Box\eta)/s = (\eta/s) \wedge \Box\eta$

Lemma 1 *Let $\sigma = s_0 s_1 \ldots s_n$ be a finite sequence of states whose length is greater than 1. Then, $M, \sigma \models \eta$ iff $M, \sigma^1 \models \eta/s_0$.*

(Proof) This lemma can be proved inductively as follows:

- $M, \sigma \models p$ iff $s \in I(s_0)$. From the definition of derivative, p/s_0 is V_T if $s \in I(s_0)$; V_F otherwise. Therefore, $M, \sigma \models p$ iff $M, \sigma^1 \models p/s_0$.

- $M,\sigma \models \eta_1 \vee \eta_2$ iff $M,\sigma \models \eta_1$ or $M,\sigma \models \eta_2$. By the induction hypothesis, this is equivalent to $M,\sigma^1 \models (\eta_1/s_0) \vee (\eta_2/s_0)$. This is equivalent to $M,\sigma^1 \models (\eta_1 \vee \eta_2)/s_0$.
- $M,\sigma \models \neg\eta$ iff $M,\sigma \not\models \eta$. By the induction hypothesis, this is equivalent to $M,\sigma^1 \not\models \eta/s_0$. This is equivalent to $M,\sigma^1 \models (\neg\eta)/s_0$.
- $M,\sigma \models \bigcirc\eta$ iff $M,\sigma^1 \models \eta$. This is equivalent to $M,\sigma^1 \models (\bigcirc\eta)/s_0$.
- $M,\sigma \models \eta_1 : \eta_2$ iff there exist $\sigma_1, \sigma_2 \in \Sigma^\dagger$ such that $\sigma = \sigma_1\sigma_2$, $M,\sigma_1 \models \eta_1$, and $M,\sigma_2 \models \eta_2$. If $|\sigma_1| = 1$, this is equivalent to $M,s_0 \models \eta_1$ and $M,\sigma^1 \models \eta_2$. If $|\sigma_1| > 1$ on the other hand, it is equivalent to $M,\sigma_1^1 \models \eta_1/s_0$ and $M,\sigma_2 \models \eta_2$. Then these conditions can be stated as follows: If $M,s_0 \models \eta_1$, it is equivalent to $M,\sigma^1 \models \eta_2 \vee ((\eta_1/s_0) : \eta_2)$; otherwise, it is equivalent to $M,\sigma^1 \models (\eta_1/s_0) : \eta_2$. Therefore, it is equivalent to $M,\sigma^1 \models (\eta_1 : \eta_2)/s_0$.
- $M,\sigma \models \boxed{:}\eta$ is equivalent to $M,\sigma \models \eta \vee (\eta : \boxed{:}\eta)$. Then, if $M,s_0 \models \eta$, it is equivalent to $M,\sigma^1 \models (\eta/s_0) \vee \boxed{:}\eta \vee ((\eta/s_0) : \boxed{:}\eta)$; otherwise it is equivalent to $M,\sigma^1 \models (\eta/s_0) \vee ((\eta/s_0) : \boxed{:}\eta)$. Therefore, it is equivalent to $M,\sigma^1 \models (\boxed{:}\eta)/s_0$. (q.e.d.)

Lemma 2 *Let $M = (\Sigma, I)$ be a linear model of ϵ-free RTL and let $\tau = s_0 s_1 s_2 \ldots$ be either finite or infinite sequence on Σ. The necessary and sufficient condition that ϵ-free RTL formula η holds along some finite prefix sequence of τ is that $M,s_0 \models \eta$ or η/s_0 holds along some finite prefix sequence of $\tau^1 = s_1 s_2 \ldots$.*

(Proof) *Necessity*: Let us assume that $M,\sigma \models \eta$, where σ is a finite prefix sequence of τ. If $|\sigma| = 1$, $M,s_0 \models \eta$. Otherwise, $M,\sigma^1 \models (\eta/s_0)$ and σ^1 is a finite prefix sequence of τ^1.
Sufficiency: If $M,s_0 \models \eta$, s_0 is clearly a finite prefix sequence of τ. If η/s_0 holds along some finite prefix sequence σ^1 of τ^1, $M,s_0\sigma^1 \models \eta$ from Lemma 1, and $s_0\sigma^1$ is clearly a finite prefix sequence of τ. (q.e.d.)

Theorem 2 *Let $K = (\Sigma, I, R, \Sigma_0)$ and η be a structure model and an ϵ-free RTL formula respectively. The necessary and sufficient condition that η is (K,s)-true at a state $s \in \Sigma$ is $s \models \eta$ or there exists a state s' such that $(s,s') \in R$ and η/s is (K,s')-true.*

(Proof) From the definition of a structure model, η is (K,s)-true if and only if there exists a finite path π on K starting from s such that $\pi \models \eta$. This is equivalent to the condition such that η holds along a finite prefix sequence π of some infinite path τ on K starting from s. Then from Lemma 2, it is equivalent to the condition that $s \models \eta$ or there exist a state s' such that $(s,s') \in R$ and η/s is (K,s')-true. (q.e.d.)

Lemma 3 *Let s be a state in a linear model $M = (\Sigma, I)$. Whether $M,s \models \eta$ or not can be decided in time $O(|\eta|)$, where $|\eta|$ is the number of operators and atomic propositions in η.*

(Proof) It can be decided inductively as follows by executing each induction step at most $|\eta|$ times:

- $M,s \models p$ iff $p \in I(s)$.
- $M,s \models \neg\eta$ iff $M,s \not\models \eta$.
- $M,s \not\models \eta_1 : \eta_2$ holds always.
- $M,s \models \eta_1 \vee \eta_2$ iff $M,s \models \eta_1$ or $M,s \models \eta_2$.
- $M,s \not\models \bigcirc\eta$ holds always.
- $M,s \models \boxed{:}\eta$ iff $M,s \models \eta$. (q.e.d.)

Theorem 3 *η/s can be obtained in time $O(|\eta|)$.*

(Proof) Derivative η/s can be obtained inductively based on its definition. In the case of $\eta = \eta_1 : \eta_2$ or $\eta = \boxed{:}\eta_1$, we need to check whether $s \models \eta_1$ holds or not. This can be also done inductively as shown in the proof of Lemma 3 and each of this induction steps can be combined with that of induction steps in the calculation of derivative. Although η_1/s appears twice in the derivative of $\boxed{:}\eta_1$, we only need to calculate it once. Therefore, each induction step can be calculated in a constant time and the number of induction step which will be executed is $|\eta|$. Thus, η/s can be calculated in time $O(|\eta|)$. (q.e.d.)

Let $D^*(\eta)$ be a set of all derivatives of η. More precisely, $D^*(\eta) \stackrel{\text{def}}{=} \{\xi | \xi = \eta/\sigma, \sigma \in \Sigma^*\}$. In the definition of $D^*(\eta)$ we regard two formulas are identical if they can be transformed to one another by using commutative, associative, and idempotence law of '$\vee$'.

Theorem 4 *For any ϵ-free RTL formula η, $D^*(\eta)$ is a finite set.*

(Proof) $D^*(\eta)$ can be shown to be a finite set inductively as follows:

- $D^*(p) = \{p, V_T, V_F\}$.
- $D^*(\neg\eta) = \{\neg\xi | \xi \in D^*(\eta)\}$.
- $D^*(\eta_1 \vee \eta_2) \subseteq \{\xi_1 \vee \xi_2 | \xi_1 \in D^*(\eta_1), \xi_2 \in D^*(\eta_2)\}$.
- $D^*(\bigcirc\eta) = \{\bigcirc\eta\} \cup D^*(\eta)$.
- Let $E_1 = \{\bigvee[\Theta] | \Theta \in 2^{D^*(\eta_2)}\}$ and $E_2 = \{\xi_1 : \eta_2 | \xi_1 \in D^*(\eta_1)\}$.
  Then, $D^*(\eta_1 : \eta_2) \subseteq \{\nu_1 \vee \nu_2 | \nu_1 \in E_1, \nu_2 \in E_2\}$
- Let $F_1 = \{\bigvee[\Theta] | \Theta \in 2^{D^*(\eta) \cup \{\boxed{:}\eta\}}\}$ and $F_2 = \{\xi_1 : \boxed{:}\eta | \xi_1 \in D^*(\eta)\}$.
  Then, $D^*(\boxed{:}\eta) \subseteq \{\nu_1 \vee \nu_2 | \nu_1 \in F_1, \nu_2 \in F_2\}$,

where $\bigvee[\Theta]$ denotes Boolean disjunction of all formulas in Θ. From the induction hypothesis, right hand sides of the above equations are finite sets. (q.e.d.)

5. Model Checking Algorithm

From Theorem 2 we obtain a model checking algorithm which decide whether an ϵ-free RTL formula η is K-true for a structure model $K = (\Sigma, I, R, \Sigma_0)$ as shown in Figure 5. It checks the condition stated in Theorem 2 by depth first search.

The procedure Addlabel(s, η, x) registers a label x which shows that the value of η is x at state s. x = 'T' means that η is (K, s)-true. x = 'F' means that η is (K, s)-false. x = 'C' means that the truth value of η at state s is now under investigation. The procedure Label(s, η) returns the label of η at s if already registered; otherwise it returns *null*. The procedure Initlabel(K, AP) registers labels for each atomic proposition, V_T, and V_F at each state. The procedure Check(K, s, η) is called only if no label is registered for η at s and checks whether η is (K, s)-true. First, it calculate the

```
procedure Verify(K, η, AP)
  Initlabel(K, AP);
  for all states s in Σ0 do {
    if Label(s, η) = 'T' then return 'T';
    if Label(s, η) ≠ 'F' then
      if Check(K, s, η) = 'T' then return 'T';
  }
  return 'F';
end of procedure

procedure Initlabel(K, AP)
  for all states s in Σ do {
    for all atomic propositions p in AP do {
      if p ∈ I(s) then Addlabel(s, p,'T');
      else Addlabel(s, p,'F');
    }
    Addlabel(s, V_T,'T');
    Addlabel(s, V_F,'F');
  }
end of procedure
```

```
procedure Check(K, s, η)
  (x, ξ) = Derivation(s, η);
  if x = 'T' then {
    Addlabel(s, η,'T');
    return 'T';
  } else {
    Addlabel(s, η,'C');
    for all s' such that (s, s') ∈ R do {
      x = Label(s', ξ);
      if x = 'T' then return('T');
      if x = 'null' then
        if Check(K, s', ξ) = 'T' then {
          Addlabel(s, η,'T');
          return 'T';
        }
    }
    Addlabel(s, η,'F');
    return 'F';
  }
end of procedure
```

Figure 5: Model Checking Algorithm

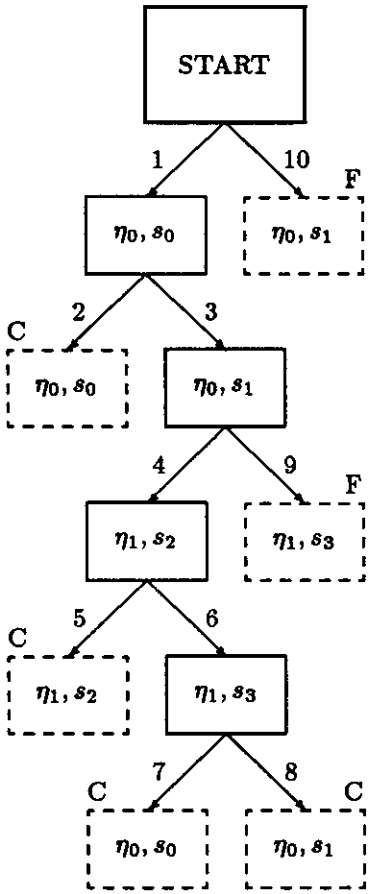
truth value of η along s and the derivative of η by s by the procedure Derivation(s,η). If $s \not\models \eta$, then it checks η/s at each successor s' of s successively. If η/s has been already labeled as 'T' at s', it returns 'T'. If η/s has been already labeled as 'F' or 'C', it proceeds to check other successors. The label 'C' means that there arises a loop in checking the truth value of η/s at s' and is treated like the label 'F' because η/s never holds along such a loop (see the proof of Theorem 5). If η/s has no label at s', it calculate the truth value of η/s at s' by calling Check$(K, s', \eta/s)$ recursively.

Theorem 5 *The procedure Verify(K,η,AP) correctly calculate K-truth value of η.*

(Proof) Its correctness is almost clear from the definition of K-truth value and Theorem 2 except its termination and loop handling. Note that Check(K,s,η) never generates derivatives of a same formula at a same state twice. Theorem 4 guarantees that only finite number of derivatives are generated by successive derivations. Therefore, it always terminates.

When Check(K,s,η) detects a loop (i.e. Label(s',ξ) = 'C', $\xi = \eta/s$), it just treats it as 'F' without checking its successors anymore because ξ never holds along any finite prefix subsequence on the loop (i.e. $\sigma^*\sigma_p \not\models \xi$ where σ is a sequence of states which constructs the loop and σ_p is any prefix subsequence of σ). This can be proved as follows:
Because Label(s',ξ) = 'C', $\xi/\sigma = \xi$ and $\sigma_p \not\models \xi$ for any prefix sequence σ_p of σ. Let us assume that there exists some prefix sequence of σ, denoted by $\sigma_{p'}$, such that $\sigma^\dagger\sigma_{p'} \models \xi$. Since $\xi/\sigma = \xi$, this implies that $\sigma_{p'} \models \xi$, which is a contradiction. Therefore, $\sigma^*\sigma_p \not\models \xi$ for any prefix sequence σ_p of σ.
(q.e.d.)

Figure 6 shows how the ϵ-free RTL model checking algorithm works on the structure model K_r (see Figure 4) and its specification *spec* of T flipflop. The frame boxes represent that the procedure



$$
\begin{array}{lcl}
\eta & \stackrel{\text{def}}{=} & p_x \equiv (p_z \equiv \bigcirc \neg p_z) \\
\eta_0 & \stackrel{\text{def}}{=} & \neg spec = \neg(\neg p_z \wedge \Box(\eta \vee LEN1)) \\
\eta_1 & \stackrel{\text{def}}{=} & \neg(p_z \wedge \Box(\eta \vee LEN1)) \\
s_0 & \not\models & \eta_0 \\
s_1 & \not\models & \eta_0 \\
s_2 & \not\models & \eta_1 \\
s_3 & \not\models & \eta_1 \\
\eta_0/s_0 & = & \eta_0 \\
\eta_0/s_1 & = & \eta_1 \\
\eta_1/s_2 & = & \eta_1 \\
\eta_1/s_3 & = & \eta_0
\end{array}
$$

Figure 6: Verification of T Flipflop by the Model Checking Algorithm

Check(K, s, η) is called with arguments written in the boxes. The dashed boxes means that the associated formulas at the specified states have been already labeled with either 'C', 'F', or 'T'. The number associated with edges represent that algorithm traverses on the structure model in this order. The edges labeled 2,5,7, and 8 causes backtracking with return value 'F' because the formulas in the dashed box pointed by these edges have been labeled 'C' at the specified states. Similarly the edges labeled 9 and 10 also causes backtracking with return value 'F' because the formulas in the dashed box pointed by these edges have been already labeled 'F' at the specified states. Then the model checking algorithm terminates with return value 'F', which means that the design of T flipflop satisfies its specification.

Next, we evaluate the execution time of the model checking algorithm. Let $Len(D^*(\eta))$ be a maximum number of operators which will be generated in derivation of $\xi \in D^*(\eta)$.

Theorem 6 *The model checking algorithm runs in time* $O((|\Sigma| Len(D^*(\eta)) \log(Len(D^*(\eta))) + |R| \log(|D^*(\eta)|))|D^*(\eta)|)$, *which is linear order of* $|\Sigma| + |R|$.

(Proof) For each $\xi \in D^*(\eta)$, Check(K, s, ξ) is called at most $|\Sigma|$ times because Check(K, s, ξ) is never called twice with same arguments. In Check(K, s, ξ), $s \models \xi$ is checked and ξ/s is generated. These steps can be done basically in time $O(Len(D^*(\eta)))$ as shown in Lemma 3 and Theorem 3. In order to guarantee termination of the algorithm, however, we need to simplify derivatives by using idempotence, commutative, and associative lows of '$\vee$'. This simplification

can be done in time $O(Len(D^*(\eta))\log(Len(D^*(\eta))))$. If $s \not\models \xi$, label of ξ/s at each successor of s is checked and Check$(K, s', \xi/s)$ is called if necessary. This step can be done in time $O(\log(|D^*(\eta)|))$ and is executed at most $|R|$ times in total during Check(K, s, η) is called for each $s \in \Sigma$. Since these steps are possibly executed for each $\xi \in D^*(\eta)$, the entire algorithm requires time $O((|\Sigma|Len(D^*(\eta))\log(Len(D^*(\eta))) + |R|\log(|D^*(\eta)|))|D^*(\eta)|)$. Because $D^*(\eta)$ is independent on a structure model K, it is proportional to the size of the structure model (i.e. $|\Sigma| + |R|$). (q.e.d.)

6. Complexity of Model Checking Problem

In this section the DTM (deterministic Turing machine) space complexity of the model checking problem of ϵ-free RTL is shown to be non-elementary with respect to the length of a given ϵ-free RTL formula. In order to show this, we show that the emptiness decision problem of an extended regular expression, whose DTM space complexity is non-elementary[1], can be transformed into the model checking problem of ϵ-free RTL in elementary time.

ϵ-free Extended Regular Expression over an alphabet Σ is defined recursively as follows:

- ϕ is an ϵ-free extended regular expression which denotes an empty set.
- s is an ϵ-free extended regular expression which denotes a set $\{s\}$, where $s \in \Sigma$.
- Let R_1 and R_2 be ϵ-free extended regular expressions which denotes language L_1 and L_2 over Σ respectively. Then, $R_1 + R_2, R_1 \cdot R_2, R_1^\dagger, R_1 \cap R_2$, and $\sim R_1$ are ϵ-free extended regular expressions which denotes $L_1 \cup L_2, L_1 L_2, L_1^\dagger, L_1 \cap L_2$, and $\Sigma^\dagger - L_1$ respectively.

For any extended regular expression, we can obtain its equivalent expression in the form either $\epsilon + R$ or R in elementary time where R is an ϵ-free extended regular expression. Furthermore, we can decide if a language denoted by an extended regular expression contains null string ϵ in linear time of the length of the given extended regular expression. Therefore, the DTM space complexity of emptiness decision problem of ϵ-free extended regular expression is also non-elementary. A language denoted by an ϵ-free extended regular expression is apparently an ϵ-free regular set.

Lemma 4 *The DTM space complexity of satisfiability problem of ϵ-free RTL is non-elementary.*

(Proof) Let AP be a set of corresponding atomic propositions such that $AP = \{p_s | s \in \Sigma\}$. Let R, R_1, and R_2 be ϵ-free extended regular expressions. Let $F(R)$ be an ϵ-free RTL formula corresponding to R. $F(R)$ is constructed inductively as follows:

- $F(\phi) = V_F$
- $F(R_1 + R_2) = F(R_1) \vee F(R_2)$
- $F(R^\dagger) = \boxdot F(R)$
- $F(\sim R) = \neg F(R)$
- $F(s) = p_s \wedge LEN1$
- $F(R_1 \cdot R_2) = F(R_1) : F(R_2)$
- $F(R_1 \cap R_2) = F(R_1) \wedge F(R_2)$

We also define ϵ-free RTL formulas φ_1, φ_2, and ψ as follows:

$$\varphi_1 \stackrel{\text{def}}{=} \bigwedge_{p_a, p_b \in AP, p_a \neq p_b} (p_a \Rightarrow p_b)$$

$$\varphi_2 \stackrel{\text{def}}{=} \bigvee_{p_s \in AP} p_s$$

$$\psi \stackrel{\text{def}}{=} \Box(\varphi_1 \wedge \varphi_2) \wedge F(R)$$

φ_1 represents that any two different atomic propositions cannot become both true at a time and φ_2 represents that at least one atomic proposition is true. Then, ϵ-free extended regular expression is empty if and only if ψ is not satisfiable. ψ can be constructed in elementary time $O(|\Sigma|^2 + |R|)$, where $|R|$ denotes the number of operators in R. (q.e.d.)

Theorem 7 *The DTM space complexity of the model checking problem of ϵ-free RTL is non-elementary with respect to the length of a given ϵ-free RTL formula.*

(Proof) We prove that the satisfiability problem of any ϵ-free RTL formula η can be transformed to the model checking problem of η in elementary time. Consider a structure model $K_c = (\Sigma, I, R, \Sigma_0)$ such that $|\Sigma| = 2^{|AP'|}$, $I : \Sigma \rightarrow 2^{|AP'|}$ is bijection, $R = \Sigma \times \Sigma$, and $\Sigma_0 = \Sigma$, where AP' is a set of atomic propositions appeared in η. Then, apparently η is satisfiable if and only if η is K_c-true. The structure model K_c can be constructed in elementary time $O(2^{2|\eta|})$. (q.e.d.)

7. Implementation and Verification Examples

The model checking algorithm stated in Section 5. has been implemented as an ϵ-free RTL model checker on VAX-11/780 under 4.3 BSD UNIX operating system and it has been used for design verification of some sequential machines with practical number of states.

7.1. Implementation of ϵ-free RTL Model Checker

In the ϵ-free RTL model checker, ϵ-free RTL formulas are stored as labeled directed acyclic binary graph in usual way. Labels are associated with nodes and represent either operators or atomic propositions and immediate successors of a node represent operands of its associated operator.

ϵ-free RTL formulas and their sub-formulas are also managed as sorted tree and same sub-formulas are shared where required in order to save spaces. Furthermore, each node has two pointers: derivative pointer and complement pointer. If a derivative of some sub-formula is generated during derivation of some formula at some state, the derivative pointer of the corresponding node is set to point its derivative. When a derivative of the same sub-formula is required again, the derivative pointer is used without re-calculating derivation. The complement pointer points its complemented formula if exists. This pointer does good job in simplifying formulas such as $\neg\neg\eta = \eta$, $\neg\eta \vee \eta = V_T$, and so on.

7.2. Verification of a Traffic Controller

In order to check the efficiency of the ϵ-free RTL model checker, it has been used for design verification of a traffic controller in [2]. The traffic controller is stationed at the intersection of a two-way

$$
\begin{aligned}
len2 &\stackrel{\text{def}}{=} \bigcirc(LEN1) \\
len3 &\stackrel{\text{def}}{=} \bigcirc(len2) \\
lengt3 &\stackrel{\text{def}}{=} \neg(LEN1 \vee len2 \vee len3) \\
nocoli &\stackrel{\text{def}}{=} \Box(\neg(E\_GO \wedge (N\_GO \vee S\_GO))) \\
icn &\stackrel{\text{def}}{=} \neg((\Box(N \wedge \neg N\_GO)) : \neg N) \\
ics &\stackrel{\text{def}}{=} \neg((\Box(S \wedge \neg S\_GO)) : \neg S) \\
ice &\stackrel{\text{def}}{=} \neg((\Box(E \wedge \neg E\_GO)) : \neg E) \\
ic &\stackrel{\text{def}}{=} \Box(icn \wedge ics \wedge ice) \\
asn4 &\stackrel{\text{def}}{=} N \wedge \neg E \wedge lengt3 \\
ass4 &\stackrel{\text{def}}{=} S \wedge \neg E \wedge lengt3 \\
ase4 &\stackrel{\text{def}}{=} E \wedge \neg(N \vee E) \wedge lengt3 \\
ngoby4 &\stackrel{\text{def}}{=} N\_GO \vee \bigcirc(N\_GO \vee \bigcirc(N\_GO \vee \bigcirc N\_GO)) \\
sgoby4 &\stackrel{\text{def}}{=} S\_GO \vee \bigcirc(S\_GO \vee \bigcirc(S\_GO \vee \bigcirc S\_GO)) \\
egoby4 &\stackrel{\text{def}}{=} E\_GO \vee \bigcirc(E\_GO \vee \bigcirc(E\_GO \vee \bigcirc E\_GO)) \\
delay4 &\stackrel{\text{def}}{=} ic \Rightarrow (\Box(asn4 \Rightarrow ngoby4) \wedge \Box(ass4 \Rightarrow sgoby4) \wedge \Box(ase4 \Rightarrow egoby4)) \\
spec &\stackrel{\text{def}}{=} nocoli \wedge delay4
\end{aligned}
$$

Figure 7: Specification of Traffic Controller

highway going north and south and a one-way road going east. It has 3 input signals (N, S, and E), 3 output signals(N_GO, S_GO, and E_GO), and 5 internal signals. N(north), S(south), and E(east) represent that there is at least one car which intends to cross the intersection straight to north, south, and east respectively. N_GO, S_GO, and E_GO represents the state of traffic lights at the intersection for the corresponding direction. It is designed as Moore machines. One design, 'bad design', which has some design error, has 43 states and its corresponding structure model has 344 states while the other ('good design') has 31 states and its corresponding structure model has 248 states.

The full specification *spec* for the traffic controllers is written in ϵ-free RTL as shown in Figure 7. $len2, len3$, and $lengt3$ means that the length of a period is 2, 3, and greater than 3 respectively. *nocoli* states that traffic lights for east direction and north-south direction never become both green at a same time. *ic* represents input constraints such that once N, S, and E are asserted, they never turn off until N_GO, S_GO, and E_GO turn on respectively. $asn4, ass4$, and $ase4$ represent the situation, whose time period is greater than 3, that there is now at least one car intending to cross the intersection to the corresponding direction while there are no cars to its orthogonal directions. $nogoby4, sgoby4$, and $egoby4$ declares that the traffic lights will be green for the corresponding direction within 4 unit times including now. Therefore, *spec* specifies that the traffic lights for the orthogonal directions to each other never becomes both green at a same time and if a car arrives at the intersection and there are no cars from its orthogonal directions the traffic light for its direction becomes green within 4 unit times including now so long as input constraints are satisfied.

$$
\begin{aligned}
len2 &\stackrel{\text{def}}{=} \bigcirc LEN1 \\
lengt2 &\stackrel{\text{def}}{=} \neg(LEN1 \vee len2) \\
as1 &\stackrel{\text{def}}{=} (\Box MemReq \Rightarrow \Box\neg ActivateComparator) \wedge (\Box\neg MemReq \Rightarrow \Box\neg MemGrant) \\
as2 &\stackrel{\text{def}}{=} \Box((CpuReq \wedge \neg DmaReq) \Rightarrow \Diamond((MemFinished \wedge lengt2) \Rightarrow \bigcirc\bigcirc \neg CpuReq)) \\
as3 &\stackrel{\text{def}}{=} \Box((\neg TransferReq \wedge \bigcirc TransferReq) \Rightarrow \bigcirc(\Diamond(DeviceReady \Rightarrow (LEN1 \vee \bigcirc DmaReq)))) \\
as4 &\stackrel{\text{def}}{=} \Box(DmaReq \Rightarrow \Diamond(ActivateComparator \Rightarrow \Diamond(ComparatorSet \Rightarrow \\
&\quad (LEN1 \vee \bigcirc(DmaEnd \vee DmaCont))))) \\
as5 &\stackrel{\text{def}}{=} \neg\Diamond(ActivateComparator \wedge MemGrant) \\
as6 &\stackrel{\text{def}}{=} \Box(\neg(TransferReq \wedge \bigcirc TransferReq \wedge (DmaType \oplus \bigcirc DmaType)) \vee LEN1) \\
as7 &\stackrel{\text{def}}{=} \Box((DmaDone \wedge ComparatorSet) \Rightarrow (DmaEnd \vee \Box DmaDone \vee (\Box DmaDone : DmaEnd))) \\
as8 &\stackrel{\text{def}}{=} \Box((\neg DmaDone \wedge ComparatorSet) \Rightarrow (DmaCont \vee \Box(\neg DmaDone) \vee \\
&\quad (\Box\neg DmaDone : DmaCont))) \\
as9 &\stackrel{\text{def}}{=} \Box(\neg(\neg ActivateComparator \wedge \bigcirc ActivateComparator \wedge ComparatorSet)) \\
asall &\stackrel{\text{def}}{=} as1 \wedge as2 \wedge as3 \wedge as4 \wedge as5 \wedge as6 \wedge as7 \wedge as8 \wedge as9
\end{aligned}
$$

Figure 8: Assertions for DMA Controller

spec contains 89 operators and the ϵ-free RTL model checker found that $\neg spec$ becomes true for the bad design in 4.7 seconds by using additional 90 expression nodes (i.e. it contains some design error) and $\neg spec$ becomes false for the good design in 19 seconds by using additional 159 expression nodes (i.e. the good design satisfies the specification). These required time and space seems to be reasonable from practical point of view.

7.3. Verification of a DMA Controller

As an example of verification of sequential machines with more states, design verification of a DMA controller in [3] has been also done. The DMA controller is designed as Moore machines with 5 input signals and 15 output signals. One design (bad design), which has some design error, has 392 states and its corresponding structure model has 12544 states. The other (good design) is a corrected version and has 272 states. Its corresponding structure model has 8704 states.

Figure 8 shows the assertions for the DMA controller. *lengt2* represents a period greater than 2 unit times. The assertion *as1* denotes that if *MemReq* is always high then *ActivateComparator* is always low and if *MemReq* is always low then *MemGrant* is also always low. *as2* asserts that it is always true that if *CpuReq* is high and *DmaReq* is low then it will eventually happen that *CpuReq* becomes low in two clock after *MemFinished* is asserted. *as3* represents that if *TransferReq* becomes high then it will eventually happen that *DmaReq* will be high at next time of *DeviceReady* being high. *as4* states that if *DmaReq* is high and *ActivateComparator* and *Comparator* will be high sometime then either *DmaEnd* or *DmaCont* will be high at the next time. *as5* states that *ActivateComparator* and *MemGrant* never becomes high at a same time. *as6* means that *DmaType* never changes its value during *TransferReq* is high. *as7* and *as8* describes that if *ComparatorSet*

		DMA Controllers (5 input signals, 15 output signals)					
		Bad Design (392 states) Structure Model 12544 states			Good Design (272 states) Structure Model 8704 states		
Assertion	#Op.	Result	Time (sec.)	#Node used	Result	Time (sec.)	#Node used
as1	10	O.K.	10.7	1	O.K.	6.4	1
as2	13	O.K.	225.9	9	O.K.	141.3	9
as3	10	O.K.	74.1	6	O.K.	41.5	6
as4	9	O.K.	117.5	7	O.K.	63.8	7
as5	3	O.K.	46.8	0	O.K.	28.4	0
as6	8	O.K.	108.6	9	O.K.	63.9	9
as7	8	Fail	12.8	3	O.K.	57.1	3
as8	11	Fail	5.6	3	O.K.	56.7	3
as9	6	Fail	3.5	2	O.K.	37.8	2
asall	86	Fail	114.3	132	O.K.	1825.8	225

Table 1: Verification of DMA Controller

is high then the value of *DmaDone* never changes until *DmaEnd* or *DmaCont* becomes high. *as9* states that *ComparatorSet* is never high just before *ActivateComparator* becomes high. *asall* is a logical conjunction from *as1* to *as9*.

Table 1 shows the result of verifications. The column '#Op.' shows the number of operators contained in assertions. The column '#Nodes' shows the number of expression nodes used in the verification process except required nodes to store given assertions themselves. $as1 \sim as9$ are checked successively while *asall* is checked separately. The ϵ-free RTL model checker finds out that assertions $as7, as8, as9$, and *asall* are not satisfied by the bad design. Assertions $as1 \sim as9$ contain $3 \sim 13$ operators. Required time to verify them varies from $3.5 \sim 225.9$ seconds, which is still acceptable from practical point of view. Especially, design errors are detected much faster and it requires only $3.5 \sim 12.8$ seconds. In order to check $as1 \sim as9$, 40 expression nodes are used additionally in total.

asall is a conjunction of $as1 \sim as9$ and contains 86 operators. It takes about 30 minutes by using 225 additional expression nodes to verify that the good design satisfies *asall*, while $as1 \sim as9$ can be verified in about 8.3 minutes by using additional 40 expression nodes in total. As shown in Theorem 6 and Theorem 7, the complexity of the model checking problem of ϵ-free RTL is non-elementary with respect to the length of a given formula but is still linear order of the size of a structure model. This means that the number of derivatives and/or their length which will be generated during model checking is non-elementary with respect to the length of a given assertion. So, if an assertion can be represented as a conjunction of some sub-formulas, it is more efficient to check sub-formulas individually than to check it as one formula from both time and space's point of view.

8. Conclusion

In this paper a method for design verification of sequential machines based on model checking of ϵ-free RTL has been described. Although the model checking problem of ϵ-free RTL has been proved to be non-elementary, the proposed model checking algorithm has been shown to be linear in the size of the structure model. The ϵ-free RTL model checker has been also implemented. It is able to determine whether given specifications written in ϵ-free RTL are satisfied by a design in reasonable time for medium size of sequential machines. For more complicated specifications it may require much more time to show that there is no design errors. However, it is still useful from practical point of view because it usually detects design errors much faster if design contains some errors. If the model checker detects some design error, it is easy to obtain ϵ-free RTL formula and a computation path along which it fails to hold. Reasoning the cause of design error from these information is an interesting future problem.

Although ϵ-free RTL can treat only finite sequence of states, full specifications of any finite state synchronous machine can be described in ϵ-free RTL because it is expressively equivalent to ϵ-free regular set. In some cases such as description of input constraints, however, it sometimes becomes easier to describe specifications by using some properties over infinite sequences. Thus the extension of ϵ-free RTL so that it can handle infinite sequences is one of the important future problems.

Acknowledgment: The author would like to express his appreciation to Prof. E.M. Clarke of CMU who supported this research by providing various utilities including design data of the traffic controller and the DMA controller. He also wishes to acknowledge Prof. Yajima, who especially encouraged him to do research at CMU for a year, Dr. Takagi, Mr. Ishiura and Mr. Hamaguchi of Kyoto Univ. for their valuable discussions.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[2] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic verification of sequential circuits using temporal logic. *IEEE Transactions on Computers*, C-35(12):1035–1044, December 1986.

[3] E. M. Clarke, S. Bose, M. C. Browne, and O. Grumberg. *The Design and Verification of Finite State Hardware Controllers.* Technical Report CMU-CS-87-145, Carnegie Mellon University, July 1987.

[4] E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In *Proc. Workshop on Logic of Programs*, pages 52–71, Springer-Verlag, 1981.

[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. *Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach.* Technical Report CMU-CS-83-152, Carnegie Mellon University, 1983.

[6] M. Fujita, H. Tanaka, and T. Motooka. Verification with Prolog and temporal logic. In *Proc. 6th Int. Symp. Computer Hardware Description Language*, pages 103–114, 1983.

[7] H. Hiraishi and S. Yajima. RTL: Regular temporal logic expressively equivalent to regular set. *Journal of Information Processing Society of Japan*, 28(2):117–123, February 1987.

[8] G. E. Hughes and M. J. Creswell. *An Introduction to Modal Logic*. Methuen, London, 1977.

[9] B. Moszkowski. *Reasoning about Digital Circuits*. Technical Report STAN-CS-83-790, Stanford Univ., 1983.

[10] N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

[11] T. Uehara, T. Saito, F. Maruyama, and N. Kawato. DDL verifier and temporal logic. In *Proc. 6th Int. Symp. Computer Hardware Description Languages*, pages 91–102, 1983.

[12] M. Vardi and P. Wolper. An automata theoretic approach to automatic program verification. In *Proc. Conf. on Logic in Computer Science*, June 1986.

[13] P. Wolper. Temporal logic can be more expressive. In *Proc. of 22nd Annual Symposium on Foundations of Computer Science*, pages 340–348, 1981.

[14] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *Proc. 24th Symp. on the Foundations of Computer Science*, pages 185–194, 1983.