数理解析研究所講究録
第 695 巻 1989 年 253-262

On Design Verification between Different Levels of Abstraction Using Regular Temporal Logic

Kiyoharu Hamaguchi, Hiromi Hiraishi and Shuzo Yajima
濱口 清治　平石 裕実　矢島 脩三
Department of Information Science
Faculty of Engineering, Kyoto University

1 Introduction

The progress of VLSI technology makes it a pressing need to establish methods for verifying the correctness of logic design. In order to verify whether a designed system satisfies a specification for it, formal verification methods have been developed.

In logic design, hierarchical design methodology is adopted to manage complex logic systems. Our main concern is to develop a formal verification method applicable to hierarchical design.

We consider formal verification of sequential machines in this paper. As a language for describing specification, we adopt *infinitary regular temporal logic*(∞*RTL*)[1] which is an extension of ϵ-free RTL proposed by Hiraishi et al. [2]. While traditional temporal logic or computation tree logic(CTL) cannot characterize finite state machines[3,4], ∞RTL is powerful enough to express regular sets and ω regular sets.

In hierarchical design, specifications and implementations are often given at different levels of abstractions. For example, a specification at register transfer level (a higher level) and an implementation at gate level (a lower level) can be given. In order to verify whether the lower-level implementation satisfies the higher-level specification, we must determine some formal relation and bridge the gap between the two levels.

In this paper, we propose a formal framework based on ∞RTL to explicitly describe relations between two different levels. We regard the relation as a part of an implementation and show a verification method for a lower-level implementation (i.e., a lower-level sequential machine and a relation) and a higher-level specification described in ∞RTL.

This paper is organized as follows: Chapter 2 introduces ∞RTL. Chapter 3 discusses a formal framework for describing relations between different levels and shows a design verification method considering two different levels. Chapter 4 summarizes this paper.

2 Regular Temporal Logic

The empty word ϵ, Σ^* and Σ^+ are defined as in the usual way. An *omega word* over an alphabet Σ is an infinite-length sequence of symbols from Σ. Σ^ω is the set of all omega words over Σ. $\Sigma^\infty \stackrel{\text{def}}{=} \Sigma^* \cup \Sigma^\omega$.

The class of infinitary regular sets is the union of regular sets[5] and ω regular sets[6].

For $\sigma \in \Sigma^\infty - \{\epsilon\}$, $|\sigma|$ denotes the length of σ, i.e., the number of symbols in σ (If σ is in Σ^ω, then we denote $|\sigma| = \omega$). $\sigma(i)$ denotes the ith symbol of σ. In the case that $|\sigma| \geq i$, σ^i denotes the suffix sub-sequence of σ starting from $\sigma(i)$.

2.1 Definition of Regular Temporal Logic

Definition 1 *Syntax*

An ∞RTL formula is simply called an RTL formula. RTL formulas are defined inductively as follows. Let AP be a set of atomic propositions. If $p \in AP$, and η and ξ are RTL formulas, then so are (p), $(\neg\eta)$, $(\eta \vee \xi)$, $(\bigcirc\eta), (\eta : \xi)$ and $(\boxdot\eta)$. $\square$

Definition 2 *Model and semantics*

$M = (\Sigma, I)$ is *a linear model*, where Σ is a set of states and $I : \Sigma \to 2^{AP}$ is an interpretation function.

Let $\sigma \in \Sigma^{\infty} - \{\epsilon\}$. $M, \sigma \models \eta$ denotes that an RTL formula η holds along the sequence σ with respect to a linear model M. If there is no confusion, M is omitted like $\sigma \models \eta$. Let p be an atomic proposition, η and ξ be RTL formulas. The relation $\models$ is defined inductively as follows:

1. $\sigma \models p$ *iff* $p \in I(\sigma(1))$.
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* $|\sigma| \geq 2$ and $\sigma^2 \models \eta$.
5. $\sigma \models (\eta : \xi)$ *iff*
   there exist $\sigma_1 \in \Sigma^+$ and $\sigma_2 \in \Sigma^{\infty} - \{\epsilon\}$
   such that $\sigma = \sigma_1\sigma_2, \sigma_1 \models \eta$ and $\sigma_2 \models \xi$
   or
   $|\sigma| = \omega$ and $\sigma \models \eta$.
6. $\sigma \models (\boxdot\eta)$ *iff*
   there exist $\sigma_i \in \Sigma^+(i = 1, \ldots, m-1)$ and $\sigma_m \in \Sigma^{\infty} - \{\epsilon\}$
   such that $\sigma = \sigma_1\sigma_2 \ldots \sigma_m$ and $\sigma_i \models \eta$ for all i
   or
   there exist an infinite number of finite sequences $\sigma_i \in \Sigma^+$
   such that $\sigma = \sigma_1\sigma_2 \ldots$ and $\sigma_i \models \eta(i = 1, 2, \ldots)$

□

In the following, '$\wedge$', V_T and V_F represent *'conjunction*, *'tautology'* and *'invalid'* respectively. Unary operators have higher precedence than binary operators. If there is no ambiguity, '(' and ')' are omitted.

Finite RTL is defined as a subclass of ∞RTL, whose semantics domain is restricted to Σ^+. Finite RTL is exactly the same as ϵ-free RTL[2].

2.2 Regular Temporal Logic and Regular Sets

First, we introduce several notations. *Len* 1 holds along a set of sequences whose length is 1. '$\Diamond$'('sometime') and '$\Box$'('always') correspond to the temporal operators used traditionally in other temporal logic. *Inf* and *Fin* represent infinite sequences and finite sequences respectively.

- $Len1 \stackrel{\text{def}}{=} \neg \bigcirc V_T$.
- $\Diamond\eta \stackrel{\text{def}}{=} \eta \vee (V_T : \eta)$.
- $\Box\eta \stackrel{\text{def}}{=} \neg\Diamond\neg\eta = \eta \wedge \neg(V_T : \neg\eta)$.
- $Inf \stackrel{\text{def}}{=} (V_T : V_F)$.
- $Fin \stackrel{\text{def}}{=} \neg Inf = \neg(V_T : V_F)$.

In order to discuss the relation between ∞RTL and regular sets, we define $L\langle\Sigma, I\rangle(\eta) \stackrel{\text{def}}{=} \{\sigma | \sigma \in \Sigma^\infty - \{\epsilon\}, \sigma \models \eta\}$, $L_f\langle\Sigma, I\rangle(\eta) \stackrel{\text{def}}{=} \{\sigma | \sigma \in \Sigma^+, \sigma \models \eta\}$ and $L_\omega\langle\Sigma, I\rangle(\eta) \stackrel{\text{def}}{=} \{\sigma | \sigma \in \Sigma^\omega, \sigma \models \eta\}$.

If there is no confusion, $L(\eta)$ etc. are used, omitting $\langle\Sigma, I\rangle$.

Theorem 1 *For an arbitrary RTL formula η and an arbitrary model (Σ, I), $L\langle\Sigma, I\rangle(\eta)$ is an ϵ-free infinitary regular set. Conversely, for an arbitrary ϵ-free infinitary regular set R over Σ, we can construct an RTL formula η such that $L\langle\Sigma, I\rangle(\eta) = R$, by introducing, for each state $s \in \Sigma$, an atomic proposition p_s such that $I(s) = \{p_s\}$.*

This theorem is proved in [1].

Corollary 1 *$L_f(\eta)$ and $L_\omega(\eta)$ are an ϵ-free regular set and an omega regular set respectively.*

From the definition of $L(\eta)$, $L_f(\eta)$ and $L_\omega(\eta)$, we can see that an RTL formula η can be used to specify some property of sequences, and $L(\eta)$ is a set of the sequences that have the property.

3 Formal Verification between Two Different Levels

3.1 Formal Framework for Describing Relations between Two Different Levels

In this section, we provide a formal framework to explicitly describe relations between the two different levels. We assume two different levels, that is, a *higher level* for a specification and a *lower level* for an implementation.

As an implementation to be verified, we consider a Mealy type deterministic sequential machine M with n binary input signals $x_1, x_2, \ldots, x_n$ and m binary output signals $z_1, z_2, \ldots, z_m$. Let $M = (X, Z, S, \delta, \lambda, s_0)$ be a Mealy type 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. $s_0 \in S$ is the initial state. $\delta : 2^X \times S \rightarrow S$ is the state transition function (We assume that at least one next state is defined for each state in S). $\lambda : 2^X \times S \rightarrow 2^Z$ (We assume that the λ is defined so long as δ is defined).

A *possible input-output sequence* of the sequential machine M is an *infinite* or *finite* sequence ρ over $2^{X \cup Z}$ such that $x_i \in \rho(k)$ iff $x_1 = 1$ at the kth input and $z_j \in \rho(k)$ iff $z_j = 1$ at the kth output, where $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$ and $k = 1, 2, \ldots, |\rho|$.

We can regard the behavior of the machine as the set of all of its possible input-output sequences. Furthermore, we can identify a possible input-output sequence with a sequence of states of ∞RTL, by introducing atomic propositions p_{x_i} and p_{z_j} 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 $z_j = 1$. From Theorem 1 and Corollary 1, we can specify the behavior of the sequential machine in finite RTL or ∞RTL.

In [2], specifications are described for *finite* possible input-output sequences by using finite RTL. While finite RTL can express any behavior of sequential machines, fairness constraints[4], which are important in describing input constraints, cannot be described. In this paper, we

(1) adopt ∞RTL to describe specifications and

(2) focus our attention to only *infinite* possible input-output sequences.

When we describe a specification at the higher level, we assume that there are possible input-output sequences at the level, even if there does not exist a realized machine, and we specify the property of the sequences by an RTL formula. In describing relations between two different levels formally, we should pay attention to higher-level and lower-level sequences of states of ∞RTL. We formalize the relations as mappings from lower-level sequences to higher-level ones.

The framework for describing the relation of two state sequences of ∞RTL is formalized by the following the *transformation rule* and *abstraction mapping.* Here subscripts $_H$ and $_L$ are used to distinguish two objects which belong to the higher level and the lower level respectively.

Definition 3 *Transformation Rule*

For two given sets of atomic propositions AP_H and AP_L, $\langle \eta_L, SI \rangle$ is called a *transformation rule*, where

- η_L is a finite RTL formula,
- $SI = \bigcup_{p_H \in AP_H} \{p_H \leftarrow f_L | f_L \text{ is a lower-level (finite) RTL formula.}\}$.

η_L is called a *time marker* and $p_H \leftarrow f_L$ a *state interpreter.* □

Definition 4 *Abstraction Mapping*

For a lower-level sequence $\langle I_L, \sigma_L \rangle$ and a transformation rule $A = \langle \eta_L, SI \rangle$, it is called *transformation of σ_L by A* to obtain a higher-level sequence $\langle I_H, \sigma_H \rangle$ such that, if $s_{L1} s_{L2} \cdots s_{Li} \models \eta_L$, then $\sigma_H(i)$ is a higher-level state such that $I_H(\sigma_H(i)) \ni p_{H_\ell}$ *iff* $s_{L1} s_{L2} \cdots s_{Li} \models \xi_{L_\ell}$ for all $p_{H_\ell} \leftarrow \xi_{L_\ell} \in SI$, otherwise $\sigma_H(i) = \epsilon$. □

Let us consider the example of Figure 1; a specification is assumed to be written at the higher level, and an implementation is given at the lower level. The higher-level adder calculates the addition (mod 16) of two integers P, Q given as inputs and then, after a *higher-level* unit delay, it outputs the result. The lower-level adder *serially* adds two integers as 4-bit binary numbers. And then, after a *lower-level* unit delay, starts to output the result.

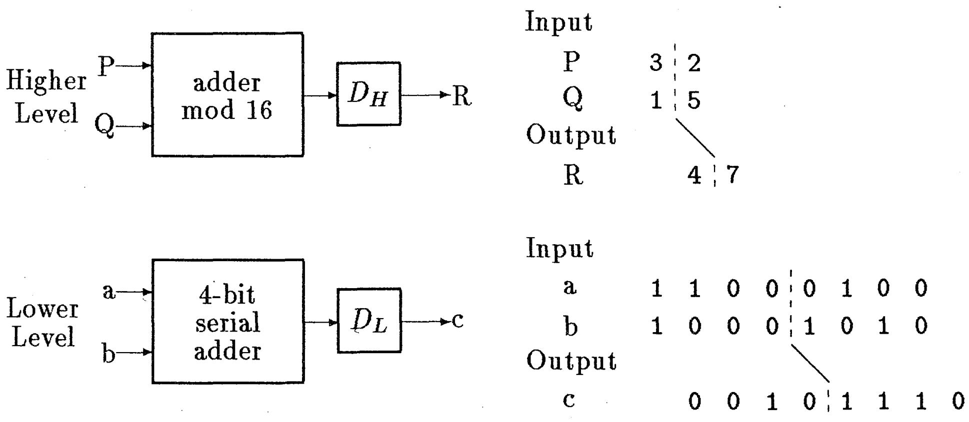

Figure 1: Adders at Two Different Levels

In Figure 1, a higher-level state corresponds to the lower-level sequences which end with four consecutive bits of inputs, and the output 0010 at the lower level corresponds to 4 at the higher level.

A transformation rule $A = \langle \eta, SI \rangle$ of the example of Figure 1 is shown as follows, where P, Q, R are represented in binary representation using atomic propositions, i.e., (p_3, p_2, p_1, p_0), (q_3, q_2, q_1, q_0), (r_3, r_2, r_1, r_0) respectively. p_3, q_3 and r_3 are the most significant bits. Here the higher-level integers are regarded as binary numbers, for simplicity.

$$
\begin{aligned}
\eta &\stackrel{\text{def}}{=} \boxed{:}\, Len4 \\
SI &\stackrel{\text{def}}{=} \{p_0 \leftarrow last(4, a), p_1 \leftarrow last(4, \bigcirc a), \cdots \\
&\quad \vdots \\
&\quad r_0 \leftarrow last(7, c), r_1 \leftarrow last(7, \bigcirc c), \\
&\quad r_2 \leftarrow last(7, \bigcirc \bigcirc c), r_3 \leftarrow last(7, \bigcirc \bigcirc \bigcirc c)\}
\end{aligned}
$$

where $last(i, \eta) \stackrel{\text{def}}{=} (\eta \wedge \ Len\ i) \vee (V_T : (\eta : Len\ i))$. $Len\ i$ holds only along the sequences that consist of exactly i states.

The example of the transformation from a lower-level sequence to a higher-level sequence of the adders of Figure 1 is shown in Figure 2.

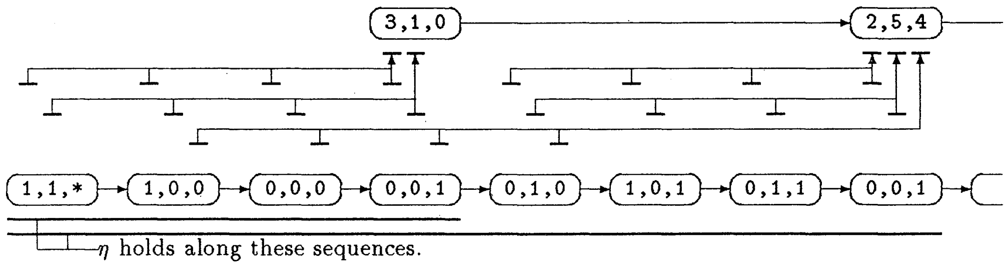

Figure 2: Transformation from a Lower-level Sequence to a Higher-level Sequence

Although the detail is omitted in this paper, we can prove that the abstraction mapping can be simulated by a *generalized sequential machine (gsm)*[5]. Because regular sets and infinitary regular sets are closed under gsm mapping[5], any higher-level sequence obtained through the abstraction mapping can be characterized by ∞RTL.

3.2 A Formal Verification Method Considering Two Different Levels

In this section, we show the outline of a formal verification method for an implementation and a specification given at two different levels.

We regard that a transformation rule is a part of an implementation.

Here a *structure model* is introduced to handle possible input-output sequences easily.

Definition 5 *Structure model*

$K = (\Sigma, I, R, \Sigma_0)$ is called a *structure model*, where (Σ, I) is a linear model of ∞RTL. $R \subseteq \Sigma \times \Sigma$ is a *total* binary relation on Σ and denotes the possible transitions between states. $\Sigma_0 \subseteq \Sigma$ is a set of initial states.

An RTL formula η is said to be *universally K-true*, if η holds along all finite and all infinite paths π from s_0 for all $s_0 \in \Sigma_0$ in the structure model K. Otherwise *universally K-false*.

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:

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

Figure 3: Generation of a Structure Model from a Sequential Machine [2]

When we focus to *only* infinite paths on the structure model K, the term *universally K-omega true (or false)* is employed. □

A structure model K corresponding to a designed sequential machine M is obtained so that the possible input-output sequences of M have one-to-one correspondence with paths on K. The ways of generating a structure model from a given Mealy machine are shown in Figure 3.

Then *formal verification* is to make sure that a given specification formula holds along all the *higher-level* state sequences obtained by the transformation rule from all the *lower-level* state sequences.

To do this, firstly, we generate a higher-level structure model K_H from the lower-level structure model K_L corresponding to the machine. The transformation is performed by applying the abstraction mapping to all the paths of K_L. Its algorithm is omitted in this paper. Our remaining work is to check whether a specification formula is universally K_H-omega true. The outline of its algorithm is shown in [7].

4 Considerations

In this paper, we show a formal framework based on ∞RTL for describing relations between two different levels of abstraction and a verification method for them.

The size of the higher-level structure model obtained from a lower-level one can be larger than that of the lower-level one. In order to avoid the increase of the size, some restriction will be necessary to the framework of abstraction mapping. However, describing the correspondence between a higher-level sequence and a lower-level one explicitly, seems a suitable approach for formal verification of hierarchical design.

Acknowledgment

The authors would like to express their sincere appreciation to Dr. N. Takagi, Mr. N. Ishiura, Mr. H. Danjo and all the members of the Yajima Laboratory in Kyoto University for their precious discussion and advice.

References

[1] K. Hamaguchi, H. Hiraishi, and S. Yajima. *A Temporal Logic Expressively Equivalent to ω-Regular Set.* Technical Report COMP88-8, IEICE, 1988. In Japanese.

[2] H. Hiraishi. *Design Verification of Sequential Machines Based on a Model Checking Algorithm of ϵ-free Regular Temporal Logic.* Technical Report CMU-CS-88-195, Carnegie Mellon University, 1989.

[3] P. Wolper. Temporal Logic Can Be More Expressive. In *Proceedings of 22nd Annual Symposium on Foundations of Computer Science*, pages 340–348, 1981.

[4] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Practical Approach. In *10th ACM Symposium on Principles of Programming Languages*, pages 117–126, January 1983.

[5] J. E. Hopcroft and J. D. Ullman. *Formal Languages and their Relation to Automata.* Addison-Wesley, 1968.

[6] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, 1976.

[7] 濱口、平石、矢島. 正則時相論理による論理設計の形式的検証. 情報基礎理論セミナー、p.121-122, 1989.