**特別研究報告書**

# 並列分解に基づく順序機械の正則時相論理による形式的検証

**指導教官　　矢島脩三 教授**

京都大学工学部情報工学科

仙石浩明

平成 2 年 2 月 15 日

# 並列分解に基づく順序機械の<br>正則時相論理による形式的検証

仙石浩明

## 内容梗概

近年の集積回路技術の発達により、設計される論理回路は複雑かつ大規模なものとなり、設計段階において誤りが生じる可能性が高まっている。その結果、設計されたシステムに誤りがない事を保証するための論理設計検証手法の確立が重要な課題となり、数学的に設計の正しさを保証する事ができる形式的検証法が重要視されている。

形式的検証は、設計する論理回路の期待される動作を規定する仕様記述と、実際に実現された論理回路の動作をモデル化したものとを用いて、モデルが仕様記述を充足する事を保証する事により行う。まず、順序機械の仕様を正則時相論理を用いて記述し、設計した順序機械の状態遷移の系列をしらみ潰し的に探索して、仕様記述から逸脱した状態遷移がないかを調べる事により、形式的に検証を行なう方法について述べる。この方法をモデル検査と呼ぶ事にする。そして、このモデル検査のアルゴリズム、その正当性、およびその計算量などを示す。

モデル検査では全ての状態遷移について調べるため、順序機械の状態数が大きいと、検査に極めて長い時間と多くの記憶容量が必要になる。しかしながら、いくつかのモジュールから構成される順序機械は、モジュール一つ一つの状態数は少なくても、全体の状態数は一般に個々の状態数の積になり、比較的少ない個数のモジュールからなる順序機械でも、その状態数は膨大なものとなる。したがって、モデル検査の最大の課題は状態数が多い順序機械をいかに扱うかということになる。

一般に状態数が多い順序機械は、中央演算処理装置やマイクロコンピュータなどに端的に現れているように、多機能かつ複数の動作モードを

持ち、各々の機能あるいはモードは他と粗な依存関係しか持たない事が多い。したがって、仕様記述も互いに粗な依存関係しか持たない複数の部分に分かれる。もしこの時、依存関係が全くないのであれば、その部分で順序機械を切り離して、別々に検査を行えば良い。これはモジュール毎に検証を行う事に相当し、大規模なシステムを作る際には一般的に使われている手法である。

しかしながら、各モジュールがバラバラに動作するだけのシステムはまれであり、その動作同士がある関係になければならない事が普通である。したがってその仕様記述は、モジュールの動作の仕様を $\eta$、$\xi$ とし、動作同士の関係を演算子 '$\circ$' で表すとすれば $\eta \circ \xi$ と表すことができるであろう。

そこで本報告では、与えられた正則時相論理式が $\eta \circ \xi$ の形である時に、これを $\eta$ と $\xi$ に分け、順序機械も $\eta$ に関係する部分機械と $\xi$ に関係する部分機械に並列分解して、$\eta$ と $\xi$ をそれぞれ対応する部分機械の上で同時にかつ別々にモデル検査することにより、$\eta \circ \xi$ のモデル検査を、少ない計算量および記憶領域で行なう方法を提案する。

例として '$\circ$' が、論理和や正則時相論理の二項演算子である場合のアルゴリズムを示し、その正当性および計算量について考察する。本手法により、特に論理和の場合は、一般に部分機械の状態数の和に比例する計算量および記憶領域でモデル検査が行なうことができる。

# Formal Verification of Sequential Machines Based on Their Decompositions Using Regular Temporal Logic

Hiroaki SENGOKU

**Abstract**

According to the recent progress of VLSI technology, more intricate and larger logic circuits are designed. The possibility that these designs contain errors becomes greater. As a result, it becomes important to establish methods for verifying correctness of logic design. Formal verification methods are strongly desired because they can guarantee correctness of design mathematically.

In order to verify correctness formaly, we use a specification and model of the behavior of implemented logic circuits, and make sure whether the model satisfies the specification. We use regular temporal logic to describe a specification, and verify correctness of design formally by checking whether all of the state transitions satisfy the specification. This formal verification method is called model checking. We discuss an algorithm of model checking, its soundness and complexity.

When the number of states of a finite state machine is large, very long time and very large storage is required to verify its correctness, because all state transitions should be basically checked in model checking. However, the number of states of a machine which consists of some modules may be, in general, the product of the number of states of each module. Thus, a machine with relatively small number of modules may have far too many states to be handled even on current super computers. Therefore, how to verify machines with many states is one

of the most significant problems in model checking.

A finite machine with large number of states has, in general, many functions and some operational modes like a CPU or a micro computer, and the funcions or the modes depend a little on each other. Thus, the specifications of these machines may be also separated into several parts, each of which corresponds to each function.

In this report, we provide a model checking algorithm based on parallel decomposition of sequential machines. We consider the specification '$\eta \circ \xi$' which is separated into two specifications '$\eta$' and '$\xi$'. Then, we decompose the machine into two submachines that correspond to '$\eta$' and '$\xi$' respectively. When the number of the states of these submachines is small, we can verify the specification '$\eta \circ \xi$' by verifying each specification on the submachines simultaneously.

We describe the algorithms in the cases that '$\circ$' is logical 'or' or concatination operation in regular temporal logic, and show its soundness and its complexity. In this method, especially in the case of logical 'or', we can model check with time and space proportional to the sum of the states of each decomposed machines.

**並列分解に基づく順序機械の**
**正則時相論理による形式的検証**

# 目次

# 1　緒論

近年の集積回路技術の発達により、設計される論理回路は複雑かつ大規模なものとなり、設計段階において誤りが生じる可能性が高まっている。その結果、設計されたシステムに誤りがない事を保証するための論理設計検証手法の確立が重要な課題となっている。従来用いられてきた論理シミュレーションによる手法では、設計誤りのない事を厳密に保証する事は困難である。

そこで数学的に設計の正しさを保証する事ができる形式的検証法が重要視されている。形式的検証は、設計する論理回路の期待される動作を規定する仕様記述と、実際に実現された論理回路の動作をモデル化したものとを用いて、モデルが仕様記述を充足する事を保証する事により行うので、この為の論理体系が必要である。

設計検証のための論理体系として、時間の概念を陽に表現できる時相論理 [1] が注目されており、分岐時間モデルに基づく CTL や、線形時間モデルに基づく命題時相論理などを用いて、実用的な設計検証システムの研究開発も試みられている [2, 3, 4, 5, 6]。

しかしながら、これらのシステムで用いられている時相論理は表現能力が低く任意の有限オートマトンの仕様を入出力線の間の関係として、記述することができないので、本報告では、有限オートマトンと表現等価な正則時相論理 [7] を用いる事にする。正則時相論理を用いた形式的検証には、設計した順序機械の状態遷移を正則時相論理で記述し充足可能性判定を行なう方法 [8] と、仕様記述から逸脱した状態遷移がないかをしらみ潰し的に探索する方法 [9] とがある。

本報告では後者の方法をモデル検査と呼ぶ事にし、この方法について考察する。このモデル検査の最大の課題は状態数が多い順序機械をいかに扱うかにある。なぜなら、状態数が大きいと、すべての状態遷移について調

べるため、検査に極めて長い時間と多くの記憶容量が必要になるからである。

状態数を減らすために、モジュール間の通信を仮想的な順序機械で置き換える方法 [10] や、同じモジュールがいくつもある場合にその閉包をとる方法 [11] などの提案がなされているが、いずれも分岐時間モデルに基づいている。本報告では、線形時間モデルに基づく状態数の削減の方法について考察する。

状態数が多い順序機械は、中央演算処理装置やマイクロコンピュータなどに端的に現れているように、多機能かつ複数の動作モードを持ち、各々の機能あるいはモードは他と粗な依存関係しか持たない事が多い。したがって、仕様記述も互いに粗な依存関係しか持たない複数の部分に分かれる。もしこの時、依存関係が全くないのであれば、その部分で順序機械を切り離して、別々に検査を行えば良い。これはモジュール毎に検証を行う事に相当し、大規模なシステムを作る際には一般的に使われている手法である。

しかしながら、各モジュールがバラバラに動作するだけのシステムはまれであり、その動作同士がある関係になければならない事が普通である。したがって、仕様記述も複数のモジュールにまたがったものになる。そこで本報告では、仕様記述を各モジュール毎の仕様と、その間の演算に分けて扱う方法を考える。そして、仕様記述のモジュール毎の仕様の分割にしたがって順序機械を並列分解し、複数のモジュールを同時に検査する事により、その相互の動作関係 (演算) も検査する手法について提案する。

この方法により、検査に必要な記憶容量および計算量は一般に小さくなる。つまり、仕様記述に現れる複数のモジュールをすべてまとめて一つの順序機械とみなして検査を行うと、その状態数は一般に各モジュールの状態数の積になるが、この手法においては同時に検査するモジュールの状態

数の和の状態数を持つ順序機械を検査する程度の時間と記憶容量ですむ。

以下、2章では本報告で用いる、正則時相論理による形式的検証についての定義および、モデル検査のアルゴリズムを述べる。

3章では順序機械を、仕様記述に基づいてどの様なモジュールに分割するかについて述べる。

そして4章で複数のモジュール上で検査を行うアルゴリズムについて述べる。

# 2 正則時相論理による形式的検証

本章では、設計された順序機械が仕様記述通りの動作をするか検査する手法を説明する。まず、2.1節で設計された順序機械の動作をどの様にモデル化するか述べる。次に、2.2節で仕様を記述する時相論理式の定義を述べる。そして、2.3節でモデルが仕様記述を充足するかどうかを検査するモデル検査法のアルゴリズムについて述べる。最後に、2.3.3節でこのアルゴリズムの計算量及び、必要な記憶領域の大きさの評価を行う。

## 2.1 仕様と設計の記述

検証を行なう対象の順序機械は決定性の Mealy 型の完全指定順序機械 $M = \langle X, Z, S, \delta, \lambda, s_0 \rangle$ とする。Moore 型の順序機械を検証する場合は、まず Mealy 型に変換してから行うものとする。ここで、

- $X$ は二値の入力信号線の有限集合。
- $Z$ は二値の出力信号線の有限集合。
- $S$ は状態の有限集合。
- $\delta : 2^X \times S \to S$ は次状態関数。

- $\lambda : 2^X \times S \rightarrow 2^Z$ は出力関数。
- $s_0 \in S$ は初期状態。

である。この順序機械からモデル $\langle \Sigma, I \rangle$ を定義する。ただし、

- $\Sigma \subseteq 2^X \times S$ : 次状態関数及び出力関数の定義域。
- $I(t)$ : $t \in \Sigma$ において、値が 1 である入力信号線 x の原始命題 $p_x$ および値が 1 である出力信号線 z の原始命題 $p_z$ の集合。

すなわち、$\Sigma$ は順序機械のすべての状態遷移の集合であり、$I$ は、各々の状態遷移で 1 の値を持つ入出力信号線がどれであるかを、原始命題の形で与える。

さて、このように定義すると、順序機械の動作は $\Sigma$ 上の言語 $\Sigma^*$ で表す事ができる。従って、仕様記述を $\Sigma$ 上の言語の部分集合として表現する事が可能になる。この場合、順序機械のすべての動作が仕様記述の集合の中に含まれるならば、その設計が正しいという事が保証される。つまり、モデル検査とは仕様記述から逸脱するような順序機械の動作がないか調べるものである。

## 2.2 正則時相論理

本節では順序機械の許される動作系列を記述する方法の一つである、正則時相論理 [7](Regular Temporal Logic, 以下 RTL と略記する) の定義を述べる。

**定義 1 (RTL の構文)** $AP$ を原始命題の集合、$p \in AP$、$\eta, \xi$ を $RTL$ 式とする時 $p$, $(\neg\eta)$, $(\eta \vee \xi)$, $(\bigcirc\eta)$, $(\boxed{:}\eta)$, $(\eta{:}\xi)$ は $RTL$ 式であり、これを有限回適応して得られるもののみが $RTL$ 式である。

**定義 2 (RTL の意味付け)** *2.1*節で述べたモデル $\langle \Sigma, I \rangle$ 上で $RTL$ の意味付けが行われる。

$\sigma^i \models \eta$ は、$RTL$ 式 $\eta$ が状態遷移 $t_i \in \Sigma$ から始まる有限長の動作系列 $\sigma^i = t_i, t_{i+1}, t_{i+2}, \ldots$ に対して真である事を意味する。言い替えれば、動作系列が仕様通りであるという事である。

- $\sigma^i \models p$ **iff** $p \in I(t_i)$
- $\sigma^i \models (\neg\eta)$ **iff** $\sigma^i \not\models \eta$
- $\sigma^i \models (\eta \vee \xi)$ **iff** $\sigma^i \models \eta$ または $\sigma^i \models \xi$
- $\sigma^i \models (\bigcirc\eta)$ **iff** $|\sigma^i| \geq 2$ かつ $\sigma^{i+1} \models \eta$
- $\sigma^i \models (\eta{:}\xi)$ **iff** $\sigma^i = \sigma_1\sigma_2, \sigma_1 \models \eta, \sigma_2 \models \xi$ となる系列 $\sigma_1, \sigma_2$ が存在する。
- $\sigma^i \models (\boxdot\eta)$ **iff** $\sigma^i = \sigma_1\sigma_2 \ldots \sigma_m, \sigma_j \models \eta (1 \leq \forall j \leq m)$ となる系列 $\sigma_j$ が存在する。

ただし、$|\sigma^i|$ は動作系列の長さである。

次のように恒真式、恒偽式を定義する。

$$V_T \stackrel{\triangle}{=} (p \vee \neg p),\ V_F \stackrel{\triangle}{=} \neg(p \vee \neg p)$$

## 2.3 モデル検査法

**定義 3 (順序機械に対する真偽値)** 順序機械 $M$ の状態 $s$ からの有限長の動作系列 $\sigma$ が存在して、$\sigma \models \eta$ となるとき、$RTL$ 式 $\eta$ は $\langle M, s \rangle$ のもとで真 *($\langle M, s \rangle$-true)* であるといい、$\sigma \models \eta$ となる $\sigma$ が存在しないとき偽 *($\langle M, s \rangle$-false)* であるという。また、初期状態 $s_0$ に対して $\eta$ が $\langle M, s_0 \rangle$-*true* である時、$M$ のもとで真、$\langle M, s \rangle$-*false* である時、$M$ のもとで偽であるという。

仕様の正則時相論理式の否定が順序機械 $M$ のもとで偽である事が示されれば、その順序機械は有限長の任意の動作系列において仕様の正則時相論理式を充足する。従って仕様通りである事が保証できるので、モデル検査は順序機械に対する真偽値を求めれば良い。

モデル検査のアルゴリズムを述べるために、まず正則時相論理式の微分について説明する。

### 2.3.1 正則時相論理式の微分

RTL 式 $\eta$ の $t \in \Sigma$ による微分を $\eta/t$ で表す。$\eta/t$ は現状態 $s$ からの動作系列において $\eta$ が成立するために次状態 (状態 $s$ から状態遷移 $t$ で到達する状態) からの動作系列で成り立つべき必要十分条件を時相論理式で表現したものである。 2.2節で述べた RTL 式の意味付けから、微分演算は次のように定義される。

$$
\begin{aligned}
p/t &\triangleq \begin{cases} V_T & p \in I(t) \text{ の場合} \\ V_F & \text{それ以外の場合} \end{cases} \\
(\neg\eta)/t &\triangleq \neg(\eta/t) \\
(\eta \vee \xi)/t &\triangleq (\eta/t) \vee (\xi/t) \\
(\bigcirc\eta)/t &\triangleq \eta \\
(\eta{:}\xi)/t &\triangleq \begin{cases} \xi \vee ((\eta/t){:}\xi) & t \models \eta \text{ の場合} \\ (\eta/t){:}\xi & \text{それ以外の場合} \end{cases} \\
(\boxed{:}\eta)/t &\triangleq \begin{cases} (\eta/t) \vee ((\eta/t){:}\boxed{:}\eta) \vee \boxed{:}\eta & t \models \eta \text{ の場合} \\ (\eta/t) \vee ((\eta/t){:}\boxed{:}\eta) & \text{それ以外の場合} \end{cases}
\end{aligned}
$$

また、特に $V_T/t = V_T$, $V_F/t = V_F$ となる。

動作系列 $\sigma = t_1, t_2, \ldots, t_n$ による微分は

$$\eta/\sigma \triangleq ((\ldots((\eta/t_1)/t_2)\ldots)/t_{n-1})/t_n)$$

と定義する。

### 2.3.2 アルゴリズム

前節で述べた微分を利用して、順序機械の状態遷移の “木” を深さ優先探索する事により、モデル検査を行う事ができる。すなわち、RTL 式 $\eta$ を充足するような “パス” が見つかった場合は真となり、見つからなかった場合は偽となる。

アルゴリズムを次に示す。

入力: 順序機械 $M$ と RTL 式 $\eta$

出力: $\eta$ が順序機械 $M$ のもとで真なら true、そうでないなら false。

**アルゴリズム 1 (モデル検査)**

**procedure** *Verify*$(M, \eta)$
**begin**
  *return Check*$(s_0, M, \eta)$
**end**

**procedure** *Check*$(s, M, \eta)$
**begin**
  **if** $\eta = V_T$ **then return true**;
  **if** $\eta = V_F$ **then return false**;
  **if** *Marked*$(M, (\eta, s))$ **then return false**;
  *Mark*$(M, (\eta, s))$;
  **for all** $t$ **such that** $\exists x, t = (x, s) \in \Sigma$ **do begin**
    **if** $t \models \eta$ **then return true**;
    **if** *Check*$(\delta(t), M, \eta/t)$ **then return true**;
  **end**;
  **return false**;

**end**;

ここで、モデル検査する対象の順序機械毎に、 RTL 式と状態の組 $(\eta, s)$ の集合を記憶する領域を用意するものとする。 $\mathrm{Mark}(M, (\eta, s))$; は順序機械 $M$ の組の集合に $(\eta, s)$ をつけ加える。 $\mathrm{Marked}(s, M, \eta)$; は順序機械 $M$ の組の集合に $(\eta, s)$ が含まれているならば真を返し、そうでないならば偽を返す。 $\mathrm{Check}(s, M, \eta)$; は $\langle M, s\rangle$-true ならば真を返し、 $\langle M, s\rangle$-false ならば偽を返す。

**for all** t **such that** $\exists x, t = (x, s) \in \Sigma$ は、状態 $s$ からのすべての状態遷移 $t$ に対して、という意味である。 $t \models \eta$ は、長さ 1 の動作系列 $t$ に対して $\eta$ が真である事を意味する。ここで注意すべき事は、 $t \models \eta$ であっても必ずしも $\eta/t = V_T$ ではないという点である。どちらも $t$ から始まる動作系列に対して $\eta$ が真という意味だが、前者は動作系列の長さが 1 であるのに対して、後者は 2 以上の動作系列に対してである。例えば RTL 式 $\xi = \neg\bigcirc V_T$ は任意の $t$ に対して、 $t \models \xi$ だが、 $\xi/t = V_F$ である。

$\delta$ は次状態関数であり、 $\delta(t)$ は状態遷移 $t$ によって到達する (次) 状態である。

### 2.3.3 正当性と評価

RTL 式 $\eta$ を繰り返し微分したときに得られる異なる式の数は高々有限個である [9] ので $\mathrm{Mark}(M, (\eta, s))$ によって登録される RTL 式と状態の組の数は有限個である。したがって、いつかは $\mathrm{Marked}(M, (\eta, s))$ が真を返すのでアルゴリズム 1は必ず停止する。

**定理 1 (モデル検査の正当性)** $\eta$ が $\langle M, s\rangle$-*true* となる必要十分条件は *Check*$(s, M, \eta)$ が真を返す事である。

**証明：** まず、$\mathrm{Check}(s, M, \eta)$ が真を返したとすると、アルゴリズム 1より明らかに $\eta/t_1t_2\cdots t_{n-1} = V_T$ または $t_n \models \eta/(t_1t_2\cdots t_{n-1})$ となる $s$ からの有限長の動作系列 $\sigma = t_1t_2\cdots t_n$ が存在する。

$\eta/t_1t_2\cdots t_{n-1} = V_T$ の場合は $t_1t_2\cdots t_{n-1}$ で到達する状態 $\delta(t_{n-1})$ からの任意の状態遷移 $t_n$ に対して $t_n \models V_T$ だから、$t_n \models \eta/t_1t_2\cdots t_{n-1}$。

故に、いずれの場合も微分の定義を繰り返し用いれば $\sigma \models \eta$ だから、$\eta$ は $\langle M, s\rangle$-true。

逆に、$\eta$ が $\langle M, s\rangle$-true だとすると、$s$ からの有限長の動作系列 $\sigma = t_1t_2\cdots t_n$ が存在して、$\sigma \models \eta$。

ここで次のような RTL 式と状態の組の列を考える。

$$(\eta, s), (\eta/t_1, \delta(t_1)), (\eta/t_1t_2, \delta(t_2)), \ldots, (\eta/t_1t_2\cdots t_n, \delta(t_n))$$

この列においては隣あう組は明らかに次の性質を持つ。

隣あう組を $(\eta_i, s_i), (\eta_{i+1}, s_{i+1})$ とすると、$s_i$ から $s_{i+1}$ への状態遷移 $t$ が存在して $\eta_i/t = \eta_{i+1}$ である。

さて、この列の中に同じもの $(\eta_i, s_i)$ が現れる場合は、二つの $(\eta_i, s_i)$ とそのあいだに挟まれる 0 個以上の組の列を一つの $(\eta_i, s_i)$ で置き換える。この操作を繰り返す事によって列の中に同じ組が現れないようにする事ができる。この操作により上記の性質は保存される。すなわち、列に対応する動作系列が常に存在する。したがって、同じ組が現れないような動作系列 $\sigma'$ が存在する。

故にアルゴリズム 1が $\sigma'$ を探索するときは $\mathrm{Marked}(M, (\eta, s))$ は真にならず、また $t_n \models \eta/t_1t_2\cdots t_{n-1}$ だから、$\mathrm{Check}(s, M, \eta)$ は真を返す。

(証明終)

次にアルゴリズム 1の順序機械の大きさに関する計算量及び使用する記憶容量について考える。

Marked が呼ばれる回数、次状態関数を評価する回数、 RTL 式を微分する回数は、いずれも Check が呼び出される回数と同じか、高々 1 異なるだけである。したがって計算量は再起呼出も含めた Check の呼びだし回数に比例する。

Check は RTL 式を繰り返し微分したときに得られる式各々に対して $|\Sigma|$ 回呼び出される。故に計算量は $O(|\Sigma|) = O(|S| \times 2^{|X|})$ となる。

必要となる記憶領域の主なものは、順序機械の次状態関数を実現するための領域と、 Mark によって登録される RTL 式と状態の組を記憶する領域である。どちらも実現の方法により大きく異なってくるが、後者は RTL 式各々に対して $|S|$ 個の組が登録され得る。したがって、 $O(|S|)$ となる。

# 3 出力従属性による順序機械の並列分解

モデル検査の最大の課題は状態数が多い順序機械どう扱うかにある。状態数が大きいと、 2.3.3節で述べたように、検査に時間がかかり、必要な記憶領域も大きくなる。そこで本章では順序機械をその出力従属性によって並列分解し、複数の順序機械に分ける事により状態数を減らす方法について述べる。まず、 3.1節で RTL 式の分割について述べる。次に、 3.2節で RTL 式の分解に応じて順序機械を並列分解して、状態数を減らす方法について述べる。

## 3.1 正則時相論理式の分割

状態数が多い順序機械は、一般に各々が異なる機能を持ついくつかのモジュールから構成されている。従って、仕様記述もモジュール毎になり検証は専らモジュール毎に行われる。しかしながら、各モジュールがバラバ

ラに動作するだけのシステムはまれであり、その動作同士がある関係になければならない事が普通である。従って仕様記述も複数のモジュールにまたがったものになる。

たとえば、RTL 式 $\eta$ 及び $\xi$ が各々別のモジュールのある動作の仕様記述だとすると、それらの動作が順に起きなければならない場合は全体の仕様記述は $\eta$;$\xi$ になる。また、両方の動作が同時に起きるという場合は $\eta \wedge \xi$ になる。

しかし、仕様記述が複数のモジュールにまたがっているからといって、関係するモジュールをまとめて一つにしてしまうと状態数が大きくなってしまう。そこでモデル検査すべき RTL 式が $\eta \circ \xi$ の形であるとき、これを分割して $\eta$ と $\xi$ の二つの RTL 式にしてモデル検査を行う事にする。

### 3.2 順序機械の出力従属性による並列分解

RTL 式の分割が順序機械のモジュール分割に沿ったものであれば、そのまま 4章で述べるモデル検査を行う事ができる。しかし分割の仕方が異なっていたり、そもそも順序機械がモジュールに分かれていない場合は、RTL 式に沿った順序機械の分解をする必要がある。

RTL 式は前述したように、入力線の原始命題 $(p_x)$ 及び出力線の原始命題 $(p_z)$ と種々の演算子から構成される。順序機械の出力信号線のうち RTL 式の出力線の原始命題に現れないものがあるならば、その出力信号線は不要である。すなわち、RTL 式に現れる出力線は元の順序機械と同じ出力を出し、現れない出力線を持たない順序機械は、出力従属性による並列分解を行う事により構成する事ができる。具体的には順序機械の状態遷移表において不要な出力を don't care にした後、状態数を最小化すれば良い。

例えば、表 1のような順序機械 $M$ を考える。

表 1: 順序機械 $M$

| $M$ | 次状態, $z_\eta z_\xi$ | |
|---|---|---|
| 現状態 | $x=0$ | $x=1$ |
| $s_1$ | $s_1, 00$ | $s_2, 00$ |
| $s_2$ | $s_2, 10$ | $s_3, 00$ |
| $s_3$ | $s_3, 01$ | $s_4, 00$ |
| $s_4$ | $s_4, 10$ | $s_5, 00$ |
| $s_5$ | $s_5, 00$ | $s_6, 00$ |
| $s_6$ | $s_6, 11$ | $s_1, 00$ |

$\eta$ は出力線の原始命題のうち $p_{z_\eta}$ のみを含み、$\xi$ は $p_{z_\xi}$ のみを含むとする。$\eta$ と $\xi$ のそれぞれについて、$M$ の出力線のうち原始命題が現れないものを don't care にして順序機械 $M'_\eta$, $M'_\xi$ を作る (表 2)。

そして、それぞれの順序機械の状態数を最小化すれば良い (表 3)。このようにして順序機械 $M$ は、$M_\eta$ と $M_\xi$ に分解される。

入力線は、RTL 式に現れない場合でも、don't care にしてしまうと順序機械の動作が非決定的になってしまうので取り除く事ができない。

## 4 複数の順序機械上でのモデル検査

以下、RTL 式 $\eta \circ \xi$ をモデル検査するものとし、順序機械 $M$ は $M_\eta = \langle X, Z_\eta, S_\eta, \delta_\eta, \lambda_\eta, s_{\eta,0} \rangle$ と $M_\xi = \langle X, Z_\xi, S_\xi, \delta_\xi, \lambda_\xi, s_{\xi,0} \rangle$ に並列分解されているものとする。ただし、$Z_\eta$ は $\eta$ に現れる出力線のみの集合であり、$Z_\xi$ は $\xi$ に現れる出力線のみの集合とする。

また、順序機械 $M$ の状態 $s$ から、出力線を don't care にして状態数を最小化した時の対応する $M_\eta$ の状態 $s_\eta$ への (多対一) 写像を $s_\eta = f_\eta(s)$ で表す。また、$f_\eta$ は状態だけでなく、状態遷移、動作系列の写像にも拡張して用いる。$M_\xi$ についても同様。

**定理 2** 順序機械 $M$ の任意の状態 $s$、状態遷移 $t$、動作系列 $\sigma$ に対して、順序機械 $M_\eta$ の状態 $f_\eta(s)$、状態遷移 $f_\eta(t)$、動作系列 $f_\eta(\sigma)$ は必ず存在し、かつ一意に定まる。

また、この時 $\sigma \models \eta$ ならば、その時に限り、$f_\eta(\sigma) \models \eta$ である。

**証明 :** 順序機械 $M_\eta$ は、決定性完全指定順序機械 $M$ の出力の一部を don't care にした後、状態数を最小化したものであるから、明らかに $f_\eta(s)$ は唯一つ存在する。次に、$M$ と $M_\eta$ の入力は同じで、両方とも決

表 2: 順序機械 $M$ の出力線を don't care にする

| $M'_\eta$ | 次状態, $z_\eta z_\xi$ | |
|---|---|---|
| 現状態 | $x=0$ | $x=1$ |
| $s_1$ | $s_1, 0-$ | $s_2, 0-$ |
| $s_2$ | $s_2, 1-$ | $s_3, 0-$ |
| $s_3$ | $s_3, 0-$ | $s_4, 0-$ |
| $s_4$ | $s_4, 1-$ | $s_5, 0-$ |
| $s_5$ | $s_5, 0-$ | $s_6, 0-$ |
| $s_6$ | $s_6, 1-$ | $s_1, 0-$ |

| $M'_\xi$ | 次状態, $z_\eta z_\xi$ | |
|---|---|---|
| 現状態 | $x=0$ | $x=1$ |
| $s_1$ | $s_1, -0$ | $s_2, -0$ |
| $s_2$ | $s_2, -0$ | $s_3, -0$ |
| $s_3$ | $s_3, -1$ | $s_4, -0$ |
| $s_4$ | $s_4, -0$ | $s_5, -0$ |
| $s_5$ | $s_5, -0$ | $s_6, -0$ |
| $s_6$ | $s_6, -1$ | $s_1, -0$ |

表 3: $M$ の並列分解

| $M_\eta$ | 次状態, $z_\eta$ | |
|---|---|---|
| 現状態 | $x=0$ | $x=1$ |
| $s_{\eta_1}$ | $s_{\eta_1}, 0$ | $s_{\eta_2}, 0$ |
| $s_{\eta_2}$ | $s_{\eta_2}, 1$ | $s_{\eta_1}, 0$ |

| $M_\xi$ | 次状態, $z_\xi$ | |
|---|---|---|
| 現状態 | $x=0$ | $x=1$ |
| $s_{\xi_1}$ | $s_{\xi_1}, 0$ | $s_{\xi_2}, 0$ |
| $s_{\xi_2}$ | $s_{\xi_2}, 0$ | $s_{\xi_3}, 0$ |
| $s_{\xi_3}$ | $s_{\xi_3}, 1$ | $s_{\xi_1}, 0$ |

定性完全指定順序機械なので、$t = (x, s)$ に対して $f_\eta(t) = (x, f_\eta(s))$ が唯一つ存在し、$\delta_\eta(f_\eta(t)) = f_\eta(\delta(t))$。

故に、$\sigma = t_1 t_2 \cdots$ に対して $f_\eta(\sigma) = f_\eta(t_1) f_\eta(t_2) \cdots$ が唯一つ存在する。

また、$M_\eta$ の出力線は $M$ の出力線の部分集合であり、$\eta$ は $Z - Z_\eta$ に含まれる出力線に対応する原始命題を持たないから、$\sigma \models \eta$ ならば、その時に限り、$f_\eta(\sigma) \models \eta$ である。

(証明終)

### 4.1 '$\vee$' 演算子の場合

RTL 式 $\eta \vee \xi$ のモデル検査を考える。$\eta$ が $M_\eta$ のもとで真となるか、または $\xi$ が $M_\xi$ のもとで真となる時、その時に限り、$\eta \vee \xi$ は $M$ のもとで真となるから、アルゴリズムは次のようになる。

**アルゴリズム 2**

**procedure** *CheckOr*$(s, M_\eta, \eta, M_\xi, \xi)$
**begin**
  **if** *Check*$(f_\eta(s), M_\eta, \eta)$ **then return true**;
  **if** *Check*$(f_\xi(s), M_\xi, \xi)$ **then return true**;
  **return false**;
**end**;

**定理 3** *CheckOr*$(s, M_\eta, \eta, M_\xi, \xi)$; は $\eta \vee \xi$ が $\langle M, s\rangle$-*true* ならば真を返し、$\langle M, s\rangle$-*false* ならば偽を返す。

**証明：** まず、$\eta \vee \xi$ が $\langle M, s\rangle$-true ならば真を返す事を示す。

$\langle M, s\rangle$-true であるから、動作系列 $\sigma$ が存在して、$\sigma \models \eta \vee \xi$。‘$\vee$’ の定義より、$\sigma \models \eta$ または $\sigma \models \xi$。定理 2より $f_\eta(\sigma) \models \eta$ または $f_\xi(\sigma) \models \xi$。したがって、$\eta$ が $\langle M_\eta, f_\eta(s)\rangle$-true または $\xi$ が $\langle M_\xi, f_\xi(s)\rangle$-true。故に、定理 1より、アルゴリズム 2は真を返す。

次に、$\eta \vee \xi$ が $\langle M, s\rangle$-false ならば偽を返すことを示す。アルゴリズム 2が真を返したとすると、定理 1から $\eta$ が $\langle M_\eta, f_\eta(s)\rangle$-true または $\xi$ が $\langle M_\xi, f_\xi(s)\rangle$-true となる。

$\eta$ が $\langle M_\eta, f_\eta(s)\rangle$-true である場合を考える。$\sigma_\eta \models \eta$ となる、順序機械 $M_\eta$ の動作系列 $\sigma_\eta$ が存在する。

$M$ と $M_\eta$ の入力線は同じであるから、$f_\eta(\sigma) = \sigma_\eta$ となる順序機械の状態 $s$ から始まる動作系列 $\sigma$ は唯一つ存在して、定理 2 より、$\sigma \models \eta$ となり、$\eta \vee \xi$ が $\langle M, s\rangle$-false であることに矛盾する。

$\xi$ が $\langle M_\xi, f_\xi(s)\rangle$-true となる場合も同様。したがって、$\sigma \models \eta \vee \xi$。

(証明終)

### 4.1.1 評価

CheckOr ではモデル検査を二回しているだけなので、順序機械の大きさに対する計算量は $O(|\Sigma_\eta| + |\Sigma_\xi|) = O((|S_\eta| + |S_\xi|) \times 2^{|X|})$ である。またモデル検査に必要な記憶領域は $O(|S_\eta| + |S_\xi|)$ である。

したがって、$|S_\eta| + |S_\xi| \ll |S|$ ならば、計算量、記憶領域共に節約できる。

## 4.2 ‘:’ 演算子の場合

RTL 式 $\eta$:$\xi$ のモデル検査を考える。アルゴリズム 1は順序機械に対する真偽判定を行う為のものであったので、RTL 式を充足する動作系列を見つけた場合は、そこで探索を打ち切って真を返した。‘$\vee$’ の場合はそれで

良かったが、‘:’ の場合はこのままでは不都合である。なぜなら、一つの順序機械において充足しても他の順序機械では充足していない事は当然有り得るので、その場合は探索を続行しなければならない。

例えば、$f_\eta(\sigma) \models \eta$ となる動作系列 $\sigma$ が存在したとする。$\sigma$ によって到達する状態を $s$ とすると、次に $\xi$ をモデル検査すれば良い。もしここで $\langle M_\xi, f_\xi(s)\rangle$-true となれば $\eta{:}\xi$ は真であるが、$\langle M, s\rangle$-false である事が分かっても $\eta{:}\xi$ が偽であるとはいえない。なぜなら、$\sigma' \models \eta$ となる動作系列が存在して、$\sigma'$ によって到達する状態を $s'$ とすると、$\xi$ が $\langle M, s'\rangle$-true となるかも知れないからである。

従って、アルゴリズムは次のようになる。

**アルゴリズム 3**

**procedure** *CheckCat*$(s, M_\eta, \eta, M_\xi, \xi)$
**begin**
  **if** $\eta = V_T$ **then begin**
    **if** *Check*$(f_\xi(s), M_\xi, V_T{:}\xi)$ **then return true**;
    **return false**;
  **end**;
  **if** $\eta = V_F$ **then return false**;
  **if** *Marked*$(M_\eta, (\eta, f_\eta(s), f_\xi(s)))$ **then return false**;
  *Mark*$(M_\eta, (\eta, f_\eta(s), f_\xi(s)))$;
  **for all** $t$ **such that** $\exists x, t = (x, s) \in \Sigma$ **do begin**
    **if** $f_\eta(t) \models \eta$ **then if** *Check*$(f_\xi(\delta(t)), M_\xi, \xi)$ **then return true**;
    **if** *CheckCat*$(\delta(t), M_\eta, \eta/f_\eta(t), M_\xi, \xi)$ **then return true**
  **end**;
  **return false**;
**end**;

アルゴリズム 1 と異なり、 Mark で順序機械 $M_\eta$ に記憶されるのは ':' の左引数を繰り返し微分して得られる RTL 式と $M_\eta$ の状態と $M_\xi$ の状態の三つ組 $(\eta, s_\eta, s_\xi)$ である。

**定理 4** *CheckCat*$(s, M_\eta, \eta, M_\xi, \xi)$; は $\eta{:}\xi$ が $\langle M, s\rangle$-*true* ならば真を返し、 $\langle M, s\rangle$-*false* ならば偽を返す。

**証明：** まず、 $\eta{:}\xi$ が $\langle M, s\rangle$-true ならば真を返す事を示す。

$s$ からの有限長の動作系列 $\sigma = t_1 t_2 \cdots t_m$ が存在して、 $\sigma \models \eta{:}\xi$。 ':' 演算子の定義から $\sigma_1\sigma_2 = \sigma$ となる動作系列 $\sigma_1, \sigma_2$ が存在して、 $\sigma_1 \models \eta$ かつ $\sigma_2 \models \xi$。

$\sigma_1 = t_1 t_2 \cdots t_n,\ \sigma_2 = t_{n+1} t_{n+2} \cdots t_m$ とする。 $\sigma_1 \models \eta$ だから $f_\eta(\sigma_1) \models \eta$。

ここで次のような RTL 式と状態の組の列を考える。

$$
\begin{aligned}
&(\eta, f_\eta(s), f_\xi(s)),\\
&(\eta/t_1, f_\eta(\delta(t_1)), f_\xi(\delta(t_1))),\\
&(\eta/t_1 t_2, f_\eta(\delta(t_2)), f_\xi(\delta(t_2))),\\
&\qquad\cdots\\
&(\eta/t_1 t_2 \cdots t_n, f_\eta(\delta(t_n)), f_\xi(\delta(t_n)))
\end{aligned}
$$

定理 1 の証明と同様にして、この列に同じ三つ組が現れないようにできるから、アルゴリズム 3 が $\sigma_1$ を探索するときは $\mathrm{Marked}(M_\eta, (\eta, f_\eta(s), f_\xi(s)))$ は真にならず、やがて $\mathrm{Check}(f_\xi(\delta(t)), M_\xi, \xi)$ が呼び出される。 $\sigma_2 \models \xi$ だから、故に $\mathrm{CheckCat}(s, M_\eta, \eta, M_\xi, \xi)$ は真を返す。

なお、 $\sigma_1' = t_1 t_2 \cdots t_{n'},\ (n' < n)$ に対して、 $\eta/\sigma_1' = V_T$ となる場合は $\eta{:}\xi/\sigma_1' = V_T{:}\xi$ だから、 $\mathrm{Check}(f_\xi(s), M_\xi, V_T{:}\xi)$ を呼び出す他は同様である。

次に、$\eta{:}\xi$ が $\langle M, s\rangle$-false ならば偽を返すことを示す。アルゴリズム3が真を返したとすると、$\eta/t_1t_2\cdots t_n = V_T$ または $t_n \models \eta/(t_1t_2\cdots t_{n-1})$ となる $s$ からの有限長の動作系列 $\sigma_1 = t_1t_2\cdots t_n$ が存在する。

$\eta/\sigma_1 = V_T$ である場合は、‘:’ の微分の定義を繰り返し用いる事により、$\eta{:}\xi/\sigma_1 = V_T{:}\xi$。アルゴリズム 3 が真を返したのであるから、有限長の動作系列 $\sigma' = \sigma'_1\sigma_2$ が存在して、$\sigma'_1 \models V_T$ かつ $\sigma_2 \models \xi$。$\sigma_1\sigma'_1 \models \eta$ かつ $\sigma_2 \models \xi$ だから、‘:’ の定義より $\sigma_1\sigma'_1\sigma_2 \models \eta{:}\xi$。

$t_n \models \eta/(t_1t_2\cdots t_{n-1})$ である場合は、微分の定義を繰り返し用いる事により、$\sigma_1 \models \eta$。アルゴリズム 3 が真を返したのであるから、有限長の動作系列 $\sigma_2$ が存在して、$\sigma_2 \models \xi$。したがって、$\sigma_1\sigma_2 \models \eta{:}\xi$。

いずれの場合も、$\eta$ は $\langle M, s\rangle$-true となり矛盾する。

(証明終)

Mark で順序機械 $M_\eta$ に記憶されるのは $\eta$ を繰り返し微分して得られる RTL 式と $M_\eta$ の状態と $M_\xi$ の状態の三つ組である。アルゴリズム 1 と同様に $\eta$ を繰り返し微分して得られる RTL 式と $M_\eta$ の状態の組のみを記憶すると、どの様な不都合が生じるかを考える。

$\eta{:}\xi$ が $\langle M, s\rangle$-true であるとする。‘:’ 演算子の定義から $\sigma_1\sigma_2 = \sigma$ となる動作系列 $\sigma_1 = t_1t_2\cdots t_n$, $\sigma_2 = t_{n+1}t_{n+2}\cdots t_m$ が存在して、$\sigma_1 \models \eta$ かつ $\sigma_2 \models \xi$。証明 4 と同様に、同じ三つ組が現れない次のような列を考える。

$$
\begin{aligned}
&(\eta, f_\eta(s), f_\xi(s)),\\
&(\eta/t_1, f_\eta(\delta(t_1)), f_\xi(\delta(t_1))),\\
&(\eta/t_1t_2, f_\eta(\delta(t_2)), f_\xi(\delta(t_2))),\\
&\cdots\\
&(\eta/t_1t_2\cdots t_n, f_\eta(\delta(t_n)), f_\xi(\delta(t_n)))
\end{aligned}
$$

ここで、三つ組の第三項、つまり順序機械 $M_\xi$ の状態を削除する。する

と、一般には $M_\eta$ の状態に対して、 $M_\xi$ の複数の状態が対応するから、第三項を削除すると、組の中に同じものが生じる可能性がある。

この時、同じものがあるからといって、定理 1 の証明と同様に置き換えを行うと $\sigma_1$ によって $M_\xi$ が到達する状態 $f_\xi(\delta(t_n))$ が一般には異なってくる。したがって、 $\eta$ を繰り返し微分して得られる RTL 式と $M_\eta$ の状態との組の列に、必ず同じものが現れないようにする事はできない。

故に、 Mark で、 $\eta$ を繰り返し微分して得られる RTL 式と $M_\eta$ の状態の組のみを記憶すると、 $\sigma_1$ を探索中でも Marked が真を返す事があり得るので、アルゴリズム 3 が真を返すとは限らなくなる。

### 4.2.1 評価

$\xi$ のモデル検査に関しては 1 と同じだから、順序機械の大きさに対する計算量は $O(|\Sigma_\xi|)$、記憶領域は $O(|S_\eta|)$ である。

$\eta$ のモデル検査では、 Mark を呼ぶときに $M_\xi$ の状態も記憶する。したがって、登録され得る $(s_\eta, s_\xi)$ の組の数を $n$ とすると、 Check は $\eta$ を繰り返し微分したときに得られる式各々に対して $n \times 2^{|X|}$ 回呼び出される。 $n$ は、 $M_\eta$ と $M_\xi$ を一つの順序機械とみなしたときの状態数よりは大きくならない。したがって、 $|S|$ よりは小さい。

故に、最悪時には計算量は $\eta$ に対して $O(|S| \times 2^{|X|})$、 $\xi$ に対して $O(|S_\xi| \times 2^{|X|})$ である。モデル検査に必要な記憶領域は $\eta$ に対して $O(|S|)$ であり、 $\xi$ に対して $O(|S_\xi|)$ である。

## 5 考察

モジュールごとの仕様記述が、各々 $\eta, \xi$ であり、これらが同時に動作すべき場合の全体の仕様記述は $\eta \wedge \xi \stackrel{\triangle}{=} \neg((\neg\eta) \vee (\neg\xi))$ である。モデル検査

をする RTL 式はこの否定だから、$(\neg\eta) \vee (\neg\xi)$ であり、アルゴリズム 2 を使う事ができる。モジュールを一まとめにしてモデル検査する場合に比べて、計算量及び記憶領域の点で、かなり節約できる。

しかし、$\eta$:$\xi$ の場合は $\xi$ の部分のモデル検査においては計算量が減るものの $\eta$ の部分のモデル検査では $M_\xi$ の状態遷移も考慮しなければならないために、最悪時には $M$ でモデル検査をした場合と変わらなくなる。

さらに、$\eta \wedge \xi$ をモデル検査する場合を考えると、$(\eta \wedge \xi)/t = (\eta/t) \wedge (\xi/t)$ であるから、$\eta$ を微分する際には同時に $\xi$ も微分しなければならない。すると、4.2節と同様にして、Mark の際に $M_\xi$ の状態だけでなく、$\xi$ を繰り返し微分したものを記憶しなければならない。したがって、$M_\eta$ と $M_\xi$ をまとめて一つの順序機械とみなしてモデル検査する場合と、ほぼ同じ計算量と記憶領域を必要とする。

'$\wedge$' 演算子で効率の向上が望めないことから、一つの動作系列に同時に複数の RTL 式が関わってくる演算 ('$\eta$:$\xi$' の場合は動作系列を二つに分割できて、前半の系列に対しては $\eta$ のみが、後半の系列に対しては $\xi$ のみが関わってくるので同時にではない。)、同様に効率の向上は望めないであろう。

したがって、モジュールの仕様記述間の任意の演算に対応できるアルゴリズムを作ることは難しい。

そこで次のような方法を考える。

**アルゴリズム 4**

**procedure** *Checks*$(s, M_1, \eta_1, M_2, \eta_2, \ldots, M_n, \eta_n)$

**begin**

$p$ := すべての入力系列の集合;

**for** $i := 1$ **to** $n$ **do**

{$p$ に含まれる入力系列で $M_i$ のモデル検査を行ない、

真になり得ない事が判明した入力系列を $p$ から取り除く $\}$;

**if** $p = \phi$ **then return false**;

**return true**;

**end**;

このアルゴリズムによれば、 $\eta_1 \wedge \eta_2 \wedge \ldots \eta_n$ をモデル検査する場合は、 $p$ から、つぎつぎに $\eta_i (i = 1, 2, \ldots, n)$ が偽になる動作系列に対応する入力系列を、 $p$ から取り除いていけば良い。

しかしながら、入力系列の集合を表現する方法および、任意の演算の場合にどのように入力系列を取り除いていくかは今後の課題である。入力系列の集合を表現するには、 RTL 式を微分して得られる異なる式の個数と、順序機械の状態数の積の個数のノードを持つグラフが必要になると予想されるので、それをコンパクトに表現する方法を考える必要があると思われる。

# 6 結論

本報告では、仕様記述を RTL で記述し、順序機械のすべての動作系列が仕様記述を満たしているかをしらみ潰しで調べる方法について考察した。

順序機械の状態数が大きいと、その分モデル検査に要する計算量および記憶容量が大きくなる。そこで、 RTL 式に応じて順序機械を出力従属性によって並列分解する方法について述べた。そして、 RTL 式のトップレベルに '$\vee$' と '**:**' がある場合には、 並列分解を行なわない場合に比べて、状態数が減った分だけ計算量および記憶容量が小さくなることを示した。

特に '$\vee$' の場合のアルゴリズムは有用であると思われる。なぜなら、仕

様記述が $\eta_1 \wedge \eta_2 \wedge \ldots$ となることは、特にいくつかのアサーションを与える場合などに、良くあることであるからである。このとき、モデル検査する RTL 式は $(\neg\eta_1) \vee (\neg\eta_2) \vee \ldots$ となり、アルゴリズム 2を使うことができる。

しかしこのままのアルゴリズムでは、任意の演算に対しては応用できない。任意の演算に対して使えるアルゴリズムは今後の課題である。

## 謝辞

本研究の機会を与えて下さり、有益な御指導を賜わりました矢島脩三教授に心から感謝致します。本研究を進めるにあたり熱心に御指導、御討論頂いた平石裕実助教授に深く感謝致します。又有益なご助言、ご討論頂く高木直史博士、石浦菜岐佐助手、荻野博幸技官、濱口清治氏をはじめとする矢島研究室の皆様に感謝致します。

## 参考文献


1. N. Rescher and A. Urquhart. *Temporal Logic*. Springer-Verlag, 1971.

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. Grümberg. The design and verification of finite state hardware controllers. Technical Report CMU-CS-87-145, Carnegie Mellon University, July 1987.

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

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

7. 平石裕実、矢島脩三. 正則集合と表現等価な正則時相論理 RTL. 情報処理学会論文誌, 28(2):117–123, 1989 年 2 月.

8. 平石裕実、浜口清治、矢島脩三. 正則時相論理の充足可能性判定アルゴリズム. 情報処理学会論文誌, 30(3):366–374, 1989 年 3 月.

9. H. Hiraishi. Design verification of sequential machines based on a model checking algorithm of $\varepsilon$-free regular temporal logic. In *Computer Hardware Description Languages and their applications*, pages 249–263, June 1989.

10. E. M. Clarke, D.E.Long, and K.L.McMillan. Compositional model checking. In *4th Annual Symposium on LOGICS IN COMPUTER SCIENCE*, pages 353–362, June 1989.

11. E. M. Clarke and O. Grümberg. Avoiding the state explosion problem in temporal logic model checking algorithms. Technical Report CMU-CS-87-137, Carnegie Mellon University, July 1987.