設 計 自 動 化 77－16
（1995. 10. 20）

部分スキャンによる同期化可能な有限状態機械の合成について

四浦 洋　井上智生　増澤利光　藤原秀雄

奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県生駒市高山町 8916-5
e-mail:hirosi-y@is.aist-nara.ac.jp

あらまし

本論文では、部分スキャンを用いて有限状態機械を同期化する問題について考察する。はじめに部分スキャンを用いた有限状態機械の同期化の原理を示し、通常入力とスキャン入力を組み合わせて有限状態機械を同期化する拡張同期化系列を提案する。次に最小個のスキャン可能な状態変数で有限状態機械を同期化可能とする問題、および、スキャン可能な状態変数の個数が与えられたとき最短の拡張同期化系列を求める問題について考察する。

キーワード　テスト容易化、有限状態機械、同期化系列、部分スキャン、拡張同期化

On the Synthesis of Synchronizable Finite State Machines with Partial Scan

Hiroshi YOURA　Tomoo INOUE　Toshimitsu MASUZAWA　Hideo FUJIWARA

Graduate School of Information Science
Nara Institute of Science and Technology
8916-5, Takayamacho, Ikoma, Nara 630-01
e-mail:hirosi-y@is.aist-nara.ac.jp

Abstract

In this paper, we present an approach to synthesizing synchronizable finite state machines using partial scan. We propose an extended synchronizing sequence which consists of normal input-patterns and partial-scan input-patterns. We consider two problems of extended synchronization; to obtain the minimum number of scannable state-valiables to synchronize an FSM, and to obtain the minimum length of extended synchronizing sequence and its state assignment for an FSM, given the number of scannable state-valiables.

key words　Testability, FSM, Synchronizing sequence, Partial Scan, Extended Synchronization

1 はじめに

有限状態機械を初期状態に依存せず、ある特定の状態に遷移させることを、有限状態機械の同期化という。また、同期化のために入力される入力系列を同期化系列という。順序回路のテスト容易性として、対応する有限状態機械が短い同期化系列を有するという性質が挙げられる。一般に有限状態機械には同期化系列が存在しない場合があり、存在する場合でも系列長が非常に長くなる場合がある。同期化系列の長さが長くなればテスト系列が長くなり、テスト容易とはいえなくなる [1][2]。

N.Jiang ら [3] は有限状態機械に対して部分スキャンを行うことで、短い同期化系列を持つ順序回路を合成する方法を提案している。しかし、この方法ではスキャン操作は一回のみと仮定しているため、スキャン可能な状態変数の数や、その同期化系列の長さが必ずしも最小とは限らない。

本論文ではスキャン操作を複数回適用するものとして、通常入力とスキャン入力による拡張同期化系列を提案する。次に有限状態機械を最小個のスキャン可能な状態変数で拡張同期化する問題について考察する。さらにスキャン可能な状態変数の個数が与えられたとき最短の拡張同期化系列を求めるアルゴリズムを提案する。

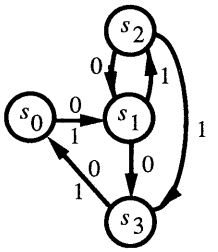

図 1: FSM M_1の状態遷移図

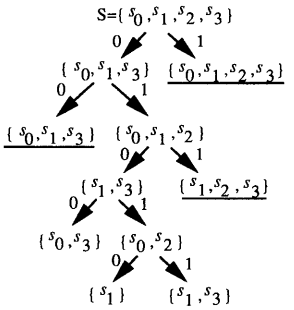

図 2: FSM M_1の同期木

2 有限状態機械の同期化

2.1 有限状態機械

本論文では有限状態機械 (Finite State Machine; 以下 FSM と記す)Mを以下のように定義する。

$$M = (S,\ I_n,\ \delta)^{\dagger}$$

ただし、Sは状態集合、I_nは入力パターン集合、δは状態遷移関数を表す。入力パターン $i_n \in I_n$に対して、δを次のように定義する。

$$s_c\ =\ \delta(i_n,\ s_p) \tag{1}$$

ここで、s_cは初期状態 s_pに入力パターン i_nを印加したときの最終状態である。入力系列$\tilde{i_n} \in I_n^+$に対する状態遷移関数$\hat{\delta}$を次のように定義する。

$$\begin{aligned} &|\tilde{i_n}| = 1 \text{ のとき}:\ \hat{\delta}(\tilde{i_n},\ s_p) = \delta(\tilde{i_n},\ s_p) \\ &|\tilde{i_n}| \geq 1 \text{ のとき}:\ \hat{\delta}(\tilde{i_n},\ s_p) = \hat{\delta}(\tilde{i'_n},\ \delta(i_n,\ s_p)) \\ &\qquad \text{ただし、} \tilde{i_n} = i_n \cdot \tilde{i'_n} (i_n \in I_n) \end{aligned} \tag{2}$$

また、状態の部分集合 $S_p \subseteq S$に対して、状態遷移関数$\hat{\hat{\delta}}$を次のように定義する。

$$\hat{\hat{\delta}}(\tilde{i_n}, S_p) = \{\hat{\delta}(\tilde{i_n},\ s_p) | s_p \in S_p\} \tag{3}$$

以下では、特に混乱のない限り、$\delta, \hat{\delta}, \hat{\hat{\delta}}$ を単にδと表す。状態の部分集合 $S_c \subseteq S$に対して、δの逆関数δ^{-1}を次のように定義する。

$$\delta^{-1}(\tilde{i_n},\ S_c) = \{s_p \in S | \delta(\tilde{i_n},\ s_p) \in S_c\} \tag{4}$$

2.2 同期化系列による FSM の同期化

FSM の同期化、および、同期化系列は次のように定義される。

[定義 1] 同期化、同期化系列 [2]
FSM Mの初期状態に依存せず、ある特定の状態へ遷移させることを Mの**同期化**という。FSM $M = (S,\ , I_n,\ \delta)$ が $\{s_t\} = \delta(\tilde{i_n},\ S)$ なる状態 $s_t \in S$と入力系列$\tilde{i_n} \in I_n^+$を持つとき、Mは同期化可能といい、$\tilde{i_n}$を Mの**同期化系列** (synchronizing sequence) という。また、同期化系列によって遷移する最終状態 s_tを**リセット状態**と呼ぶ。■

†本論文では FSM の出力については考慮しないので、FSM の出力に関する定義は省略する。

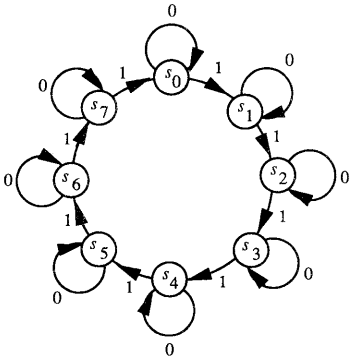

図 3: FSM M_2の状態遷移図

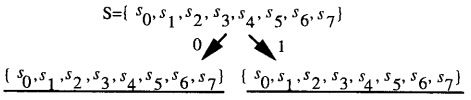

図 4: FSM M_2の同期木

与えられたFSM $M = (S,\ I_n,\ \delta)$ が同期化系列を有するかどうかを調べたり、同期化系列が存在する場合にそれを求める問題は同期木 (図 2) を作ることにより解くことができる [2]。同期木の各節は状態集合を、各枝は状態遷移を表す。根は全ての状態集合 Sを表し、節が表す状態集合を S_pとすると、各入力パターン $i_n \in I_n$について $S_c = \delta(i_n, S_p)$ となる子が作られ、それぞれの枝には対応する入力パターンを表すラベルがつけられる。同期木において、各節内の状態の数を**曖昧度** [2](ambiguity) という。

例 1: 図 1の FSM $M_1 = (S,\ I_n,\ \delta)$ の状態集合は $S = \{s_0, s_1, s_2, s_3\}$ である。M_1は S(曖昧度=4) から入力系列 01010 を印加することにより、$\{s_1\}$(曖昧度=1) となり、状態 s_1に同期化できる（図 2参照）。すなわち、$\{s_1\} = \delta(01010,\ S)$ であり、入力系列 01010 が M_1の同期化系列となる。 ■

すなわち同期化系列とは $|S| = n$ とすると、S（曖昧度 $=n$）から単一の状態からなる集合（曖昧度=1）に遷移させるような入力系列である。

図3の FSM M_2においては同期木は図4のようになり、同期化できない。次節ではこのような FSM を部分スキャンにより同期化する方法について述べる。

2.3 部分スキャンを用いた拡張同期化

本節ではまず部分スキャンについて説明し、次に部分スキャンを用いて FSM を同期化する方法について説明する。

FSM を順序回路に合成するために、状態の符号化、すなわち、状態割当てを行わなければならない。状態割当てとは、FSM の状態数を n とすると、1 個の状態を $m = \lceil \log_2 n \rceil$ 個の状態変数の値の組で表すことである。状態割当てを行い、符号化した状態のうち、$2^m - n$ 個の FSM の状態に対応しない状態が存在する。このような状態を**無効状態**という。状態変数は順序回路の記憶素子 (フリップフロップ) に対応している。部分スキャンとは、いくつかのフリップフロップを通常入力とは別に設けられたスキャン入力により 0、1 どちらの値にでも設定できるようにする設計である。この設定を**スキャン操作**と呼び、スキャン操作可能なフリップフロップに対応する状態変数を**スキャン可能な状態変数**という。また、いくつかのフリップフロップの値をリセット入力により 0、1 どちらか一方の値にのみ設定できるようにする設計もある。この設定を**リセット操作**と呼び、リセット操作可能なフリップフロップに対応する状態変数を**リセット可能な状態変数**という。

m 個の状態変数 $y_0, y_1, \cdots, y_{m-1}$を持ち、そのうちの k 個がスキャン可能な状態変数である FSM を考える。一般性を失うことなく、$y_0, y_1, \cdots, y_{k-1}$をスキャン可能な状態変数とすることができる。

FSM の状態集合を $S = \{s_t | 0 \le t \le 2^m - 1\}$ とする。Pomeranz ら [5] は部分リセットを用いる場合について、リセット可能でない状態変数による状態集合 Sの分割を考えた。ここでは部分スキャンを用いる場合について、以下の分割を定義する。スキャン可能な状態変数による S の分割として、

$$\pi_s = \{T_0, T_1, \cdots, T_{2^k-1}\}$$
$$T_i = \{s_t | y_0(s_t) y_1(s_t) \cdots y_{k-1}(s_t) = i\}$$

を考える。ここで、$y_0(s_t)y_1(s_t)\cdots y_{k-1}(s_t) = i$ とは、$y_0(s_t)y_1(s_t)\cdots y_{k-1}(s_t)$ を 2 進数とみたとき、その値が i と等しいことを表す。このときπ_sを**スキャン分割**といい、T_iを**スキャン分割ブロック**という。

同時に、スキャン可能でない状態変数による状態集合 Sの分割として、

$$\pi_{ns} = \{P_0, P_1, \cdots, P_{2^{m-k}-1}\}$$
$$P_j = \{s_t | y_k(s_t) y_{k+1}(s_t) \cdots y_{m-1}(s_t) = j\}$$

を考える。π_{ns}を**非スキャン分割**といい、そのブロックP_jを**非スキャン分割ブロック**という。

π_sとπ_{ns}の積について、次式が成り立つ。

$$\pi_s \cdot \pi_{ns} = \{\{s_0\}, \{s_1\}, \cdots, \{s_{2^m-1}\}\} = \mathbf{0} \quad (5)$$

以下では、スキャン可能な状態変数$y_0 y_1 \cdots y_{k-1}$をiに設定するスキャン操作を**スキャン入力iを印加する**という。また、スキャン入力iはi_sと書き、通常入力と区別する。次に、部分スキャン設計されたFSM Mを以下のように定義する。

$$M = (\alpha(S),\ I,\ \sigma)$$

ただし、$\alpha(S)$は全ての$s_t \in S$が状態割当てされた状態集合、$I = I_n \cup I_s$（I_n：通常入力パターン集合、I_s：スキャン入力パターン集合）、σは状態遷移関数を表す。また、スキャン可能な状態変数が与えられており、状態集合$\alpha(S)$はスキャン分割ブロック$T_i(0 \le i \le 2^k - 1)$、非スキャン分割ブロック$P_j(0 \le j \le 2^{m-k} - 1)$にそれぞれスキャン分割、非スキャン分割されているものとする。入力パターン$i \in I$に対して、σを次のように定義する。

$$\begin{aligned} s_c &= \sigma(i,\ s_p) \\ &= \begin{cases} \delta(i,\ s_p), & i \in I_n \\ T_i \cap P_j, & i \in I_s \end{cases} \end{aligned} \quad (6)$$

ここで、$s_c \in \alpha(S)$は初期状態$s_p \in P_j$に入力パターンiを印加したときの最終状態である。式(2)と同様に、入力系列$\tilde{i} \in (I_n \cup I_s)^+$に対して、$\hat{\sigma}$を以下のように定義する。

$$\begin{aligned} &|\tilde{i}| = 1 \text{のとき：}\ \hat{\sigma}(\tilde{i},\ s_p) = \sigma(\tilde{i},\ s_p) \\ &|\tilde{i}| \ge 1 \text{のとき：}\ \hat{\sigma}(\tilde{i},\ s_p) = \hat{\sigma}(\tilde{i}',\ \sigma(i,\ s_p)) \\ &\qquad \text{ただし、} \tilde{i} = i \cdot \tilde{i}'(i \in I) \end{aligned} \quad (7)$$

また、式(3)と同様に、状態の部分集合$S_p \subseteq \alpha(S)$に対して、状態遷移関数$\hat{\sigma}$を次のように定義する。

$$\hat{\sigma}(\tilde{i}, S_p) = \{\hat{\sigma}(\tilde{i},\ s_p) | s_p \in S_p\} \quad (8)$$

以下では、特に混乱のない限り、$\sigma, \hat{\sigma}, \hat{\sigma}$を単に$\sigma$と表す。式(4)と同様に、状態の部分集合$S_c \subseteq \alpha(S)$に対して、$\sigma$の逆関数$\sigma^{-1}$を次のように定義する。

$$\sigma^{-1}(\tilde{i},\ S_c) = \{s_p \in \alpha(S) | \sigma(\tilde{i},\ s_p) \in S_c\} \quad (9)$$

非スキャン分割ブロックP_jは、スキャン入力i_sにより$\{s_t\}$（$s_t \in T_i \cap P_j$）に遷移する。

$$\{s_t\} = \sigma(i_s,\ P_j),\ s_t \in T_i \cap P_j \quad (10)$$

例 2: FSM M_2に対して表1のように状態割当てを行う。スキャン可能な状態変数をy_0とすると、M_2においてスキャン分割ブロック、非スキャン分割ブロックは

表 1: FSM M_2の状態割当て

	y_2	y_1	y_0
s_0	0	0	0
s_1	1	0	0
s_2	0	1	0
s_3	1	1	0
s_4	0	0	1
s_5	1	0	1
s_6	0	1	1
s_7	1	1	1

スキャン可能な状態変数:y_0

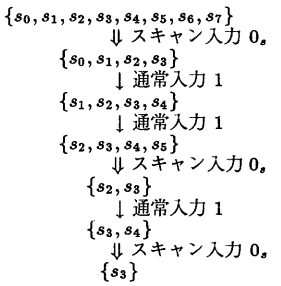

図 5: FSM M_2の拡張同期化の流れ

$$T_0 = \{s_0, s_1, s_2, s_3\},\ T_1 = \{s_4, s_5, s_6, s_7\},$$
$$P_0 = \{s_0, s_4\},\ P_1 = \{s_1, s_5\},$$
$$P_2 = \{s_2, s_6\},\ P_3 = \{s_4, s_7\}$$

である。例えば、式(6)より、状態s_0のスキャン入力0_s、1_sによる遷移は、

$$\sigma(0_s,\ s_0) = s_0,\ \sigma(1_s,\ s_0) = s_4$$

となる。また、状態集合$S_p = \{s_0, s_1, s_3, s_4, s_5\}$とすると、式(7)より、$S_p$のスキャン入力$0_s$、$1_s$による遷移は

$$\sigma(0_s,\ S_p) = \{s_0, s_1, s_3\},$$
$$\sigma(1_s,\ S_p) = \{s_4, s_5, s_7\}$$

となる。 ■

このように部分スキャン設計されたFSMは、スキャン入力を印加することにより、曖昧度を減らし、状態を特定することができる。

2.4 拡張同期化系列

通常入力とスキャン入力による同期化を特に拡張同期化と呼ぶ。以下にFSMの拡張同期化、および、拡張同期化系列を定義する。

[**定義 2**] 拡張同期化、拡張同期化系列
FSM Mの初期状態に依存せず、スキャン入力と通常入力によりある特定の状態に遷移させることを Mの**拡張同期化**という。また、Mの状態集合を Sとすると、Mが $\{s_t\} = \sigma(\tilde{i},\ S)$ なる入力系列$\tilde{i} \in (I_n \cup I_s)^+$を持つとき、$M$は拡張同期化可能といい、入力系列$\tilde{i}$を Mの**拡張同期化系列** (extended synchronizing sequence) という。 ■

例 3: 表 1のように状態割当ての行われた FSM M_2 が与えられたとする。M_2は通常入力だけからなる同期化系列を持たない（図 4参照）。そこで、M_2に対して部分スキャン設計を用いて拡張同期化する。スキャン可能な状態変数を y_0とする。M_2は入力系列 $0_s110_s10_s$により、状態 s_3に拡張同期化できる（図 5参照）。すなわち、$\{s_3\} = \sigma(0_s110_s10_s,\ S)$ であり、入力系列 $0_s110_s10_s$が拡張同期化系列となる。 ■

長さ $l(\geq 2)$ の入力系列$\tilde{i_s} = i_{1s}i_{2s}\cdots i_{ls} \in I_s^+$による状態集合 S_pの遷移は$\sigma(\tilde{i_s},\ S_p) = \sigma(i_{ls},\ S_p)$ となる。すなわち、連続するスキャン入力系列$\tilde{i_s}$を最後に印加するスキャン入力 i_{ls}に置き換えることにより、より短い拡張同期化系列が得られる。従って、以下で扱う拡張同期化系列には、スキャン入力が連続して現れることはないものとする。

2.5 拡張同期化問題

少ないハードウェアオーバヘッドでテスト容易化するためには、スキャン可能な状態変数の数を最小にすること、および、拡張同期化系列を最短にすることが重要である。これまでの例では、状態割当てが決まっている FSM を扱ってきたが、FSM に短い拡張同期化系列を持たせるためには、状態の割当て方が重要である。また、状態割当てが決まっている場合、拡張同期化可能にするためには、すべての状態変数をスキャン可能にしなければならない FSM が存在することを示すことができる。したがって、ここでは次の 3 つの拡張同期化問題について考察するが、いずれの問題も与えられる FSM は状態割当てが行われていないものとする。

[問題 1]
FSM Mが与えられたとき、最小個のスキャン可能な状態変数で Mを拡張同期化可能とするために必要な状態割当てを求める。 ■

[問題 2]
FSM Mとスキャン可能な状態変数の数 kが与えられたとき、最小個の状態付加により Mと等価な拡張同期化可能 FSM M'とその状態割当てを求める。 ■

[問題 3]
FSM Mとスキャン可能な状態変数の数 kが与えられたとき、状態付加を行わずに Mが拡張同期化系列を持つかどうか判定し、持つときには最短の拡張同期化系列とその状態割当てを求める。 ■

3 スキャン可能な状態変数の最小化

本節では、前節で定義した問題1、2について考察する。

文献 [3] でも、スキャン操作は一回のみという仮定のもとで問題 2 を考察しており、FSM M が与えられたとき、等価な状態を追加することにより、1 個のリセット可能な状態変数を用いて長さ 2 の拡張同期化系列を持つ FSM M' を構成する方法を示している。この構成法では、M （状態数を n とする）の 1 つの通常入力に着目し、その入力によって l 個の状態が同一の状態に遷移する場合、$n - 2l$個の状態を追加することにより M' を構成している。

本節では、以下の定理を示す。この定理は、スキャン操作を複数回行う場合について、同期化可能にするために必要なリセット可能な状態変数の数の上界 (問題 1)、および、k個のリセット可能な状態変数で同期化可能にするために付加する状態の数の上界 (問題 2) を与えている。

定理 1 状態数 n の状態機械の通常入力 i_nに関する状態遷移図を $G(i_n)$ とする．また, $G(i_n)$ の弱連結成分の数を $m(i_n)$ と表す．このとき, 通常入力 i_n とスキャン入力（またはリセット入力）による同期化に関して、以下が成り立つ.

(1) $G(i_n)$ に自己ループが存在しないとき

(a) $n \geq 4$, かつ, $G(i_n)$ が $n/2$ 個の長さ 2 のサイクルのみからなるとき：

- $n/2$ が偶数の場合、1 個のスキャン可能な状態変数では同期化不可能。$n/2$ が奇数の場合、1 個のリセット可能な状態変数では同期化不可能であり、1 個のスキャン可能な状態変数で同期化可能。また、いずれの場合も、2 個のリセット可能な状態変数で同期化可能。
- （等価な）状態を 1 個追加すれば, 1 個のリセット可能な状態変数で同期化可能.

(b) その他の場合：
1 個のリセット可能な状態変数で同期化可能.

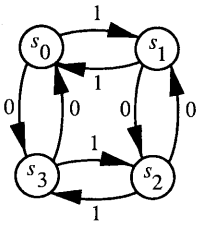

図 6: FSM M_3:同期化できない FSM の例

(2) $G(i_n)$ に自己ループが存在するとき.

- $(1-\frac{1}{2^{k-1}})n+1 < m(i_n) \leq (1-\frac{1}{2^k})n+1$ なら, $k-1$ 個のスキャン可能な状態変数では同期化不可能であり、k 個のリセット可能な状態変数で同期化可能.
- k個のスキャン可能な状態変数では同期化不可能な FSM に、状態を$\frac{2^k}{2^k-1}(m(i_n)-1)-n-1$ 個追加しても, k個のスキャン可能な状態変数では同期化不可能であり、(等価な) 状態を$\frac{2^k}{2^k-1}(m(i_n)-1)-n$ 個追加すれば, k個のリセット可能な状態変数で同期化可能。

■

(**注 1**) 定理 1 において $k=1$ とすれば、(等価な) 状態を $2(m(i_n)-1)-n$ 個追加すれば, 1 個のリセット可能な状態変数で同期化可能であることが分かる。入力 i_nによって l 個の状態が同一の状態に遷移する場合、$m(i_n) \leq n-l+1$ となり、高々$n-2l$個の状態を追加することにより 1 個のリセット可能な状態変数で同期化可能であることが分かる。このことより、同期化可能にするために付加する状態数について、定理 1 は文献 [3] の結果の一般化である。

(**注 2**) 定理 1 では、2 種類以上の入力を利用した同期化を考慮していない。従って定理 1 では、同期化可能にするために必要なスキャン可能な状態変数の数の上界を与えているが、2 種類以上の入力を考慮した場合には、より少ないスキャン可能な状態変数で、同期化できる場合もある (1 個のスキャン可能な状態変数で同期化可能にするために付加する状態変数の数についても同様)。ただし、たとえ状態遷移図に自己ループを含まないような FSM であっても、2 種類以上の通常入力を利用しても 1 個のスキャン可能な状態変数では同期化できないものが存在する。図 6はその一例である。従って、定理 1(1) で示す、同期化可能にするために使用するリセット可能な状態変数の数 (2 個) は、最悪時を考える限り、最小数である (1 個のスキャン可能な状態変数で同期化可能にするために追加する状態変数の数についても同様)。

定理 1 より、FSM が 1 個のリセット可能な状態変数で同期化可能なための十分条件は、ある通常入力 i_nについて以下が成り立つことである。

1. $G(i_n)$ に自己ループが存在しないとき、$n \leq 3$, または, $G(i_n)$ が長さ 2 のサイクル以外の弱連結成分を含む。
2. $G(i_n)$ に自己ループが存在するとき、$m(i_n) \leq n/2+1$。

つまりこの手法において、1 個のリセット可能な状態変数では同期化できないのは、どの通常入力 i_nに関しても、$G(i_n)$ が $n/2$ 個の長さ 2 のサイクルからなるか、あるいは、$G(i_n)$ が $n/2+1$ 個より多い弱連結成分からなるときのみである。MCNC'89[4] のベンチマーク回路について確かめたところ、すべての回路がこの十分条件を満たし、1 個のリセット可能な状態変数で同期化可能であることが分かった。したがって、実際の回路では、この 1 個のリセット可能な状態変数で同期化できるための十分条件が成り立つと考えられる。

4 拡張同期化系列の最短化

本節では、問題 3 について考察する。与えられた FSM Mとスキャン可能な状態変数の数 kに対して、最短の拡張同期化系列を求めるためには、(1) 系列を構成する通常入力、スキャン入力、(2) 状態割当て、(3) 無効状態の遷移、のすべての組合せの中から探索する必要がある。ここでは、効率良く短い拡張同期化系列を求めるために発見的手法を用いたアルゴリズムを提案する。

図 7に、発見的手法を用いた最短の拡張同期化系列を求めるアルゴリズムを示す。本アルゴリズムは、逆順探索法 [3] をスキャン入力に対応するように拡張したものである。はじめに逆順探索法について説明する。

逆順探索法 (Reverse Order Search;ROS)

同期木を用いて同期化系列を求める方法は、最短の同期化系列が求められるが、計算量が与えられた FSM の状態のべき集合に比例するため、大きい FSM には適用で

きない。逆順探索法は発見的手法により、同期木の逆順に曖昧度が1からnとなるまで探索し、同期化系列を求める手法である。まずリセット状態となる候補$\{s_r\}$から始める。現在の状態集合をS_cとすると、$|\delta^{-1}(i_{sct}, S_c)| = max_{i_n \in I_n}(|\delta^{-1}(i_n, S_c)|)$を満たす$i_{sct}$を選択し、$S_c$から$\delta^{-1}(i_{sct}, S_c)$に探索を移す。順次、全ての状態集合$S$となるまで探索を続け、それまでに選択した入力を逆順にたどると同期化系列となる。

例4: 1のFSM M_1の同期化系列を逆順探索法により求める。リセット状態をs_1とする。

$$\begin{aligned}
\delta^{-1}(0, \{s_1\}) &= \{s_0, s_2\} \\
\delta^{-1}(1, \{s_0, s_2\}) &= \{s_1, s_3\} \\
\delta^{-1}(0, \{s_1, s_3\}) &= \{s_0, s_1, s_2\} \\
\delta^{-1}(1, \{s_0, s_1, s_2\}) &= \{s_0, s_1, s_3\} \\
\delta^{-1}(0, \{s_0, s_1, s_3\}) &= S
\end{aligned}$$

よって同期化系列は01010となる（図2参照）。 ■

次に、本アルゴリズムの流れを説明する。

1. **リセット状態の選択：**（$cand\_reset()$）
   $max_{i_n \in I_n}(|\sigma^{-1}(i_n, s_r)|) = max_{s_t \in S}(max_{i_n \in I_n}(|\sigma^{-1}(i_n, s_t)|))$を満たす状態$s_r$をリセット状態となる候補とする。現在の状態集合$S_c$を$\{s_r\}$とする。

2. **終了条件：**
   現在の状態集合S_cが無効状態を含む全ての状態集合$S \cup S_{inv}$であれば、それまでの入力系列を逆順にたどった系列を拡張同期化系列として終了する。

3. **バックトラック条件：**
   $|S_c| \leq 1$、または、S_cがそれまでのS_cとして現れているとき（すなわち、$(0 \leq m \leq l-1) \wedge (S_c[m] = S_c[l])$を満たす$m$が存在するとき）、一つ前の$S_c$にバックトラックする（backtrack()）。

4. **入力の選択：**（$select\_input()$）
   $|\sigma^{-1}(i_{xn}, S_c)| = max_{i_n \in I_n}(|\sigma^{-1}(i_n, S_c)|)$を満たす通常入力$i_{xn}$を求める（$select\_nomal\_input()$）。また、状態集合$S'_c = \{s_t | s_t \in S_c, State\ assignment\ is\ performed\ for\ s_t\}$に対して、$|\sigma^{-1}(i_{xs}, S'_c)| = max_{i_s \in I_s}(|\sigma^{-1}(i_s, S'_c)|)$を満たすスキャン入力$i_{xs}$を求める（$select\_scan\_input()$）。$i_{xs}$による遷移を決めるため、以下の2つのステップを行う。

   (a) **無効状態の追加：**
   無効状態があれば、現在の状態集合S_cにそれを加えることで、$|\sigma^{-1}(i_{xs}, S_c)|$をより大きくすることができる。ここでは、$S_c$を一回更新した後、全ての無効状態を$S_c$に加える。また、$S_c$に無効状態を加えた時、その遷移は$S_c$内の任意の状態の遷移と同じにする（$trans\_change()$）。

   (b) **状態割当て：**（$state\_assign()$）
   状態集合$\sigma^{-1}(i_{xs}, S_c)$は、以下の状態割当てを行うことによって求められる。ここでいう状態割当てとは、状態変数の値を決めることではなく、状態がどのスキャン分割ブロック、非スキャン分割ブロックに属するか決めることである。ここでは後に自由度を残すため、状態集合$\sigma^{-1}(i_{xs}, S_c)$を求めるために必要な割当てのみが行われる。必要な状態割当てとは、S_c内の状態を全て状態割当てした後、それらの状態の属する全ての非スキャン分割ブロックの要素数が最大（2^k個）となるように状態を割り当てることである（式(10)参照）。この状態割当ても、$|\sigma^{-1}(i_{xs}, S_c)|$が最大になるも

```
min_ess(M, k)
/* M: FSM, k: number of scannable state variables */
{
    S = valid_states(M);
    S_inv = invalid_states(M);
    I_n = normal_input_patterns(M);
    I_s = scan_input_patterns(k);
    l = 0;
    s_r = cand_reset();
    S_c[l] = {s_r};
    while(S_c[l] ≠ S ∪ S_inv){
        if(|S_c| ≤ 1 and S_c is alredy tried) backtrack();
        i_sct = select_input(S_c, ess, l);
        ess[l] = i_sct;
        l++;
        S_c[l] = σ^-1(S_c[l-1], i_sct);
    }
    return reverse(ess);
}
select_input(S_c, ess, l)
{
    i_xn = select_normal_input(S_c[l]);
    if(l == 0 or ess[l-1] ∉ I_s){
        I_xs = select_scan_input(S_c[l]);
        if(l ≠ 0 and S_inv is not included){
            S_c[l] = S_c[l] ∪ S_inv;
            trans_change(S_inv);
        }
        state_assign(S_c[l]);
        if(|σ^-1(i_xn, S_c[l])| > |σ^-1(I_xs, S_c[l])|){
            i_sct = i_xn;
        }else{
            i_sct = I_xs;
        }
    }
    return i_sct;
```

図7: 最短の拡張同期化系列を求めるアルゴリズム

のが選ばれる。また、S_cの要素以外の状態割当てされる状態は、$S_p = \sigma^{-1}(i_{xs}, S_c)$とすると、$max_{i_n \in I_n}(\sigma^{-1}(i_n, S_p))$が最大になるように選択する。

$|\sigma^{-1}(i_{xn}, S_c)|$と$|\sigma^{-1}(i_{xs}, S_c)|$を比較し、大きい方の入力を$i_{sct}$として選択する。

5. 現在の状態集合$S_c = \sigma^{-1}(i_{sct}, S_c)$として、ステップ2へ。

例5: 図8のFSM M_3、スキャン可能な状態変数の数1が与えられたとする。無効状態をs_7とする。はじめにスキャン分割ブロックT_0, T_1、非スキャン分割ブロックP_0, P_1, P_2, P_3を用意する。

(1) $\sigma^{-1}(1, s_1) = \{s_0, s_6\}$、$\sigma^{-1}(0, s_3) = \{s_3, s_4\}$、また、$max_{s_t \in S - \{s_1, s_3\}}(max_{i_n \in I_n}(|\sigma^{-1}(i_n, s_t)|)) \leq 1$であるから、$s_1, s_3$のうち任意の状態を選択する。ここでは$s_1$を選択する。$S_c = \{s_1\}$、$ess = NILL$。

(2) $\sigma^{-1}(0, S_c) = \{s_1\}$、$\sigma^{-1}(1, S_c) = \{s_0, s_6\}$、また、$max_{i_s \in I_s}(|\sigma^{-1}(i_s, S_c)|) \leq 2$である。通常入力1を選択する。$S_c = \{s_0, s_6\}$、$ess = 0$。

(3) $\sigma^{-1}(0, S_c) = \{s_0, s_6\}$、$\sigma^{-1}(1, S_c) = \{s_3, s_5\}$である。ここで$S_c$に無効状態$s_7$を加える。状態割当てを$T_0 = \{s_0, s_6, s_7\}, T_1 = \{s_i, s_j, s_k\}$ $P_0 = \{s_0, s_i\}, P_1 = \{s_6, s_j\}, P_2 = \{s_7, s_k\}$のように行うと、$\sigma^{-1}(0_s, S_c) = \{s_0, s_6, s_7, s_i, s_j, s_k\}$で最大となり、この状態割当てを選択する。$s_i, s_j, s_k$はそれぞれ$s_1, s_2, s_3$となる。そしてスキャン入力0を選択する。$S_c = \{s_0, s_1, s_2, s_3, s_6, s_7\}$、$ess = 00_s$。

(4) ここでは通常入力による遷移のみを考える。$\sigma^{-1}(0, S_c) = \{s_0, s_1, s_2, s_3, s_4, s_6\}$, $\sigma^{-1}(1, S_c) = \{s_0, s_1, s_2, s_3, s_5, s_6\}$, 通常入力1を選択する。$S_c = \{s_0, s_1, s_2, s_3, s_5, s_6\}$、$ess = 00_s1$。

(5)T_0にs_4、T_1にs_5を加え、$P_3 = \{s_4, s_5\}$とする状態割当てを行うと、$\sigma^{-1}(1_s, S_c) = S \cup S_{inv}$となる。$S_c = S \cup S_{inv}$、$ess = 00_s11_s$。

以上よりM_3の拡張同期化系列は1_s10_s0となる。 ■

5 まとめ

本論文では、部分スキャンを用いて有限状態機械を同期化する問題について考察した。通常入力とスキャン入力を組み合わせて有限状態機械を同期化する拡張同期化系列を提案し、拡張同期化問題について考察した。その結果として、有限状態機械を拡張同期化するために必要なスキャン可能な状態変数の個数に関する定理、および、スキャン可能な状態変数の個数が与えられたとき、最短の拡張同期化系列を求めるアルゴリズムを示した。今後の課題として、ベンチマーク回路に対するアルゴリズムの有効性の評価が挙げられる。

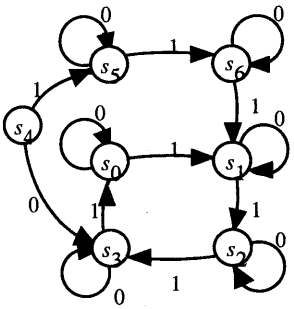

図8: FSM M_4の状態遷移図

謝辞

本研究に関し、多くの貴重な意見を頂いた高畠勝之氏始め情報論理学講座の皆様方に感謝致します。

参考文献

[1] K. Cheng and V. Agrawal, "Initializability consideration in sequential machine synthesis," *IEEE Trans. on Computers*, vol. 41, pp. 374-379, Mar 1992.

[2] A. Miczo, "The Sequential ATPG: A Theoretical Limit," *Proc. Int. Test Conf*, pp.143-147, 1983.

[3] N. Jiang, R. M. Chou and K. K. Saluja, "Synthesizing Finite State Machines for Minimum Length Synchronizing Sequence Using Partial Scan. " *FTCS*, vol. 25, Jun. 1995.

[4] S. Yang, "Logic synthesis and optimization benchmarks user guide version 30," Microelectronics Center of North Carolina, Research Triangle Park, NC, Tech. Rep, Jan. 1991.

[5] I. Pomeranz and S. Reddy, "On the Role of Hardware Reset in Synchronous Sequential Circuit Test Generation," *IEEE Trans. on Computers*, vol. 43, pp. 1100-1105, Sep 1994.

[6] J. Rho, F. Somenzi, and C. Pixley, "Minimum Length Synchronizing Sequences of Finite State Machine." *Proc. Design Automation Conf*, pp. 463-468, 1993.

[7] H. Fujiwara, "Logic Testing and Design for Testability." *The MIT Press*, 1985.