一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
SS2017-16,KBSE2017-16(2017-07)

組込みソフトウェアを対象とした状態遷移表の抽出と分析支援の検討

山本　椋太[†]　吉田　則裕[†]　青木　奈央[††]　高田　広章[†]

† 名古屋大学大学院情報学研究科　〒 464–8603 愛知県名古屋市千種区不老町
†† 組込みシステム技術協会　〒 103–0011 東京都中央区日本橋大伝馬町
E-mail: †{muku,yoshida,hiro}@ertl.jp, ††nao@zipc.com

あらまし　レガシー化した組込みソフトウェアの理解が困難になっており，保守や再利用に大きなコストがかかる．リアルタイム制御を行う組込みソフトウェアを状態遷移モデルで表現可能な場合が多く存在する．本研究では，組込みソフトウェアのレガシーコードから状態遷移表を抽出し，ソースコードの理解を支援する手法を提案する．また，状態遷移表が大規模になる可能性があるため，状態遷移表の分析を支援するための手法について検討する．
キーワード　リバースエンジニアリング，コード解析，状態遷移表

On the Extraction of State Transition Tables from an Embedded Software System and Analysis Support

Ryota YAMAMOTO[†], Norihiro YOSHIDA[†], Nao AOKI[††], and Hiroaki TAKADA[†]

† Graduate School of Infomatics, Nagoya University　Furo-cho, Chikusa-ku, Nagoya, Aichi, 464–8603 Japan
†† Japan Embedded Systems Technology Association　Odenma-cho 6–7, Chuo-ku, Tokyo, 103–0011 Japan
E-mail: †{muku,yoshida,hiro}@ertl.jp, ††nao@zipc.com

Abstract　It is hard to understand legacy code for an embedded software system. It leads much cost for maintaining and reusing the system. Real-time control of an embedded software system can be represented as a state transition model. In this paper, we propose an approach to extract state transition tables from legacy code for understanding embedded real-time systems. Also, we also discuss a method to support developers in analyzing large and complex state transition tables.
Key words　Reverse-Engineering, Code Analysis, State Transition Table

1. はじめに

組込みソフトウェア開発において，ソースコードのレガシー化により，再利用や保守のコストが増大している．担当者が既に存在せず，設計書も存在しないソースコードが，再利用や保守の活動を阻害している．この問題を軽減させるために，レガシーコードを理解するための支援が必要であると考えられる．

筆者らは，これまでにも組込みソフトウェアを対象としたリバースエンジニアリングに関する検討を進めてきた [1], [2]．しかしながら，これまでの手法では下記の問題点がある

【問題点 1】　JIS 規格等の規格に沿った表を出力することができていない．

【問題点 2】　イベントや状態の網羅性を欠いている．

【問題点 3】　「組込みシステム」の特性に対する考慮が不足しており，リアルタイム OS のアプリケーションを対象と仮定したとき，割込みハンドラや周期ハンドラはイベントになる可能性を考慮できていない．

以上の問題点を解決するべく，本稿では状態遷移表の抽出手法を改良することを目的としている．

また，状態遷移表から形式手法によるモデル (以下，形式モデルと呼ぶ) を抽出し，それを用いた分析の検討を行う．開発者が理解している仕様と，コードをリバースエンジニアリングした結果が一致しているかを分析するための支援のために形式手法を使用する．ここで対象にするソースコードは状態遷移設計されていることとするため，状態遷移系のモデルにリバースすることで，開発者の理解を支援する．コードからのリバースエンジニアリングではなく，状態遷移表からのリバースエンジニアリングによって形式モデルを抽出することの意味は，制約式を抽出することができないことにある．形式モデルにおける対象システムの仕様は，状態遷移表抽出までの中間生成物から抽出することができる．しかし，時相論理などを用いた分析については，状態遷移表を参照しながらユーザが時相論理式を定

This article is a technical report without peer review, and its polished and/or extended version may be published elsewhere.
Copyright ©2017 by IEICE

義する必要がある．このように，コードからの抽出だけではなくユーザの記述によって形式モデルが記述される．

本稿では，状態遷移表の抽出方法とそのツール化，および形式手法の利用について述べる．

2. 研究動機

これまでにも筆者らはC言語から状態遷移表を抽出する手法(以降，従来手法と呼ぶ)について研究してきた[1]．従来手法では，半自動の状態遷移表抽出ツールとしての実装を行った．状態遷移表が抽出される対象のコードの状態を表す変数を状態変数と呼んでいるが，状態変数の選択はユーザが行う．従来手法では状態変数候補を抽出することは可能であるが，状態変数候補の中から状態遷移表を表現する上でより有効な変数を抽出することは，ユーザの意図にも依存するため難しい．そこで，この手順のみ手動としている．従来手法において抽出される状態遷移表を図1に示す．しかし，従来手法には問題があることがわかったため，その改良を進めてきた．

【問題点1】として，JIS規格[3]等の規格に従っていないということがあげられ，状態遷移表の可読性が低下している．すなわち，抽出された状態遷移表を見たとき，処理と遷移の区別のために時間がかかってしまう．JIS規格では，遷移と処理を明確に区別して表記しているため，一見して処理と遷移を区別できる．さらにJIS規格とは状態遷移表の読み方が異なっている．従来手法によって抽出された図1の状態遷移表では，表を上から下に向かって読み，一度のループで複数のイベントの処理を実行するようなことが考えられる．このとき，図1のように状態とイベントの組み合わせが一意ではないことも考えられる．しかし，JIS規格では一度のループで1つのセルを参照すればよく，かつ必ず状態とイベントの組み合わせは一意になっている．

【問題点2】としては，状態とイベントの組み合わせが網羅されていないことである．図1においては，$S > X1$というイベントが存在するが，$S \leqq X1$というイベントは存在していない．すなわち，イベントと状態の網羅性が状態遷移表の特徴であるが，それを活かすことができないことになる．

【問題点3】としては，「組込みシステム」の特性に対する考慮が不足していることがあげられる．リアルタイムOSのアプリケーションを対象と仮定したとき，割込みハンドラや周期ハンドラがイベントになる可能性を考慮できていない．

上記の問題点を解決するべく，新しい手法(以降，提案手法と呼ぶ)を検討した．

3. 提案手法とツールの実装

3.1 提案手法

状態遷移表抽出の過程を図2に示す[2]．この手順は以下のとおりである．

(手順1)　ソースコードからイベント処理表を抽出

(手順2)　シーケンス表から状態変数を選択

(手順3)　選択した状態変数に関する状態遷移表を抽出

まず，(手順1)について，イベント処理表の概要を説明する．

条件		state		
		S1	S2	ELSE
無条件実行		t = in1; s = in2;	t = in1; s = in2;	t = in1; s = in2;
t==1	無条件実行	S++;	S++;	S++;
	S<X1	out = S;	out = 0; state = S3;	S++; state = S1
無条件実行		return 0;	return 0;	return 0;

図1　導出された状態遷移表

イベント処理表とは，ある条件の組み合わせにおいて実行される処理を関数ごとにまとめた表である．本稿においては，状態遷移表を抽出するための中間状態としてイベント処理表を作成する．イベント処理表の作成例を図2(a)に示す．イベント処理表は，イベント部と処理部に分割される．ここで，処理文の列Pおよび条件文の列Cを考えると，以下のようになる．ただし，Cについて，elseが存在する場合にはそのelseを条件式に展開している．

$$\begin{aligned} P =& [\text{“}int\ t = in1, s = in2;\text{”}, \text{“}S++;\text{”}, \text{“}out = S;\text{”}, \\ & \text{“}out = 0;\text{”}, \text{“}state = S3;\text{”}, \text{“}state = S1;\text{”}] \end{aligned} \tag{1}$$

$$\begin{aligned} C =& [\text{“}t == 1\text{”}, \text{“}state == S1\text{”}, \text{“}state == S2\text{”}, \\ & \text{“}!(state == S1)\&\&!(state == S1)\text{”}, \text{“}out == 0\text{”}] \end{aligned} \tag{2}$$

ここで，if-else構文でCを分割すると，以下の通りになる．もしelseに対応する条件式が存在しない場合，その条件式を追加している．

$$C_1 = \{\text{“}t == 1\text{”}, \text{“}!(t == 1)\text{”}\} \tag{3}$$

$$\begin{aligned} C_2 =& \{\text{“}state == S1\text{”}, \text{“}state == S2\text{”}, \\ & \text{“}!(state == S1)\&\&!(state == S1)\text{”}\} \end{aligned} \tag{4}$$

$$C_{2_3} = \{\text{“}out == 0\text{”}, \text{“}!(out == 0)\text{”}\} \tag{5}$$

ただしC_{2_3}はC_2の3つめの条件式であるelseに包含されている条件式の集合である．C_iの直積をとることで，図2(a)のようにイベントの集合を求めている．

ここから，選択した状態変数に対する状態遷移表を導出する．状態変数の定義は，以下のとおりである．

- 状態変数は分岐条件に使用されている．
- 状態変数が取りうる値は有限個である．
- 状態変数は内部で更新される．

本手法では，状態変数を1つだけ選択する場合を考える．いま，図2(b)のように状態変数$v = state$としたとき，状態値の集合Vは下記の通りになる．

$$V = \{\text{“}S1\text{”}, \text{“}S2\text{”}, \text{“}!(S1)\&\&!(S2)\text{”}\} \tag{6}$$

このときvを含む条件式を削除し，Vを状態遷移表の状態列と

して表現することで，図 2(c) のように状態遷移表の列が表現されている．

同じく，状態遷移表におけるイベントの導出に関しては，すべての条件式の集合から v を含む条件式を削除し，残りの条件式を状態遷移表におけるイベントとしており，図 2(c) 中の行として表現されている．

最後に，イベントと状態の組み合わせに一致する処理を P から当てはめることで図 2(d) のような状態遷移表を導出できる．

また，割込みの扱いについて検討する．割込みハンドラに関しては，イベントの 1 つとして扱うことができる [4]．そのため，状態遷移表の末尾に行を追加し，割込みハンドラ内に状態変数を含む条件分岐がある場合には，さらに状態に合わせてセルに処理を書き込む．

以上のようにして状態遷移表を作成することができる．

3.2　ツールの実装

ツール化においては，従来手法を利用する [1]．従来手法では，ある関数における条件と処理の対応表である条件処理表をソースコードから作成している．その過程で，条件式のネストの深さと条件と別の条件や処理の依存関係を抽出している．これを活用して制御フローを抽出する．

制御フローから，条件シーケンスの全てのパターンとそれに対応する処理を得ることができる．これがイベント処理表である．イベント処理表は，以下のとおりである．

以降，イベント処理表の導出の流れを説明する．例として図 3 を使用する．この図においては $ とラベルの付いたノードが存在するが，これは後述する「無条件」を表している．ここでは条件式のシーケンスをイベントと呼び，イベント列の導出について説明する．これらの過程では，条件式をそのまま扱わず，各条件式に自然数 ID を割り当て，ID をもとに進めていく．

Seq_E の導出では，はじめに条件式をネストの深さごとに分ける．そして，ネストの深さごとに条件式に else が存在するかどうかを判定し，もし else が存在しなければ else を補完する．このとき補完された else の中では特に何も処理は行われない．ここで，従来手法ではすべての処理のネストの深さを揃えるために「無条件」と呼ばれる if($true$) と等価である条件式を追加していることに注意し，「無条件」の条件式には else を補完しない．これらの処理によって，図 4 が導出される．

ここから，条件式の依存関係を使用してイベントを作成していくが，そのために図 5 のような依存関係が求められていることとする．この図においても $ は「無条件」を表している．ここでは，処理を考慮せずに進めていくためすべてのノードは条件式である．まず，条件式の依存関係から，子となる条件式を持たないノードをすべて求める．これらは，制御フローにおける葉ノードとなる．葉ノードから根ノードまでをたどることで，1 つのイベントとしている．ただし，同じネストの深さの条件式かつ if-if の構造になるような場合には，各 if-elseif-else でひとまとまりのパスを求め，それぞれの直積をとることでパスとすることができる．上記のような対応をとることが可能であるが，現状では「無条件」のみ if-if の構造が生じた場合を考慮した実装としている．

```
if(state == A){
        if(sw == OFF){
                state = B;
                power = OFF;
        }
        if(system == BUSY)
                led = ON;
}
else if(state == B){
        sleep(100);
}
```

⇒

条件文リスト	処理

state == 1
⇒ sw == 0
⇒ system == 5
	state = 2;
	power = 0;
	led = 1;

state == 1 ⇒ sw == 0 ⇒ !(system == 5)	state = 2; power = 0;
state == 1 ⇒ !(sw == 0) ⇒ system == 5	led = 1;
state == 1 ⇒ !(sw == 0) ⇒ !(system == 5)	##NONE##
state == 2	sleep(100);
!(state == 1) && !(state ==2)	##NONE##

(a) 入力ソースコードのイベント処理表への変換

条件文リスト	処理

state == 1
⇒ sw == 0
⇒ system == 5
	state = 2;
	power = 0;
	led = 1;

state == 1 ⇒ sw == 0 ⇒ !(system == 5)	state = 2; power = 0;
state == 1 ⇒ !(sw == 0) ⇒ system == 5	led = 1;
state == 1 ⇒ !(sw == 0) ⇒ !(system == 5)	##NONE##
state == 2	sleep(100);
!(state == 1) && !(state ==2)	##NONE##

(b) 状態変数の選択

条件文リスト	処理

state == 1
⇒ sw == 0
⇒ system == 5
	state = 2;
	power = 0;
	led = 1;

state == 1 ⇒ sw == 0 ⇒ !(system == 5)	state = 2; power = 0;
state == 1 ⇒ !(sw == 0) ⇒ system == 5	led = 1;
state == 1 ⇒ !(sw == 0) ⇒ !(system == 5)	##NONE##
state == 2	sleep(100);
!(state == 1) && !(state ==2)	##NONE##

⇒

	state		
	1	2	!1 && !2
sw == 0 ⇒ system == 5			
sw == 0 ⇒ !(system == 5)			
!(sw == 0) ⇒ system == 5			
!(sw == 0) ⇒ !(system == 5)			

(c) イベントと状態値の抽出

	state		
	1	2	!1 && !2

sw == 0
⇒ system == 5
	state = 2;
	power = 0;
	led = 1;
		sleep(100);	##NONE##

sw == 0 ⇒ !(system == 5)	state = 2; power = 0;	sleep(100);	##NONE##
!(sw == 0) ⇒ system == 5	led = 1;	sleep(100);	##NONE##
!(sw == 0) ⇒ !(system == 5)	##NONE##	sleep(100);	##NONE##

(d) 処理の抽出

図 2　状態遷移表抽出の流れ

このようにして，図 5 を図 6 のようなツリーのように変換し，結果としてイベントの集合を得ることができる．

最後に，対応する処理列の導出にあっては，すでに従来手法

```
void task(){                     // f1
  if(state == A){                // c1
    if(sw == OFF){               // c2
      state = B;                 // p1
      power = OFF;               // p2
    }
    if(true){                    // $
      led = ON;                  // p3
    }
  }
  else if(state == B){           // c3
    if(true){                    // $
      sleep(100);                // p4
    }
  }
}
```

図 3　対象ソースコード

ネストの深さ	条件式ノード
0	f_1 ○
1	c_1 ● c_3 ● e_1 ●
2	c_2 ● e_2 ● $ ● $ ●

○ :関数
● :条件文
● :条件文(無条件)
● :補完されたelse

図 4　else が補完された条件式の階層

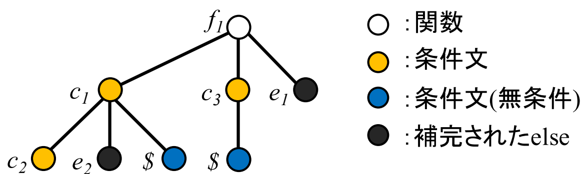

図 5　導出された状態遷移表

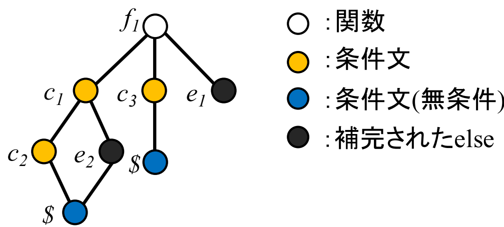

図 6　導出された状態遷移表

によって，ある条件式に対応する処理を抽出できている．そのため，イベントを構成する条件式が持つ処理を列として抽出するだけでよい．

以上をイベント処理表の抽出ツールとして実装した．

続いて，状態変数の選択を行う．また，状態変数の選択についても従来手法を利用する．従来手法では，状態変数の定義を全て満たしている変数である状態変数候補を抽出し，ユーザに提示する対話型 UI がある．その UI を活用して，ユーザが手作業で状態変数を選択する．

最後に，選択した状態変数名をユーザがツールに渡し，状態遷移表を作成する．作成する状態遷移表は，ユーザから指定された関数についてのものとする．状態遷移表の抽出は以下の手順からなるが，現在割込みハンドラについて実装を検討している．

- 割込みハンドラをイベント化する
- 選択された状態変数に関する状態列を作成する
- 既存のイベントの条件式列から状態変数を削除する
- 状態変数の変更に関わる処理を遷移処理として視覚的に理解しやすいようにする
- 対応するセルに処理を書き込む

まず，はじめに割込みハンドラのイベント化を検討する．ここまでの操作で，割込みハンドラもひとつの関数であるから，割込みハンドラに対するイベント処理表が導出されている．そこで，割込みハンドラのイベントの冒頭に関数名を付与し，割込みハンドラを構成要素とした新しいイベントを作成する．そして，そのイベント集合を対象関数のイベント集合に加える．以上によって割込みハンドラに対応できると考えている．

次に，対象関数のイベント集合について，選択された状態変数がとりうる値を抽出する．このとき，取りうる値はイベントから抽出する．こうして抽出した値を列見出しとして，状態列を作成する．その後，状態変数を含む条件式をイベント集合から削除する．そうすると，重複したシーケンスを持つイベントがイベント列内に生じうるため，プロセスに書き込む状態列の情報を付加しておき，イベントはマージしておくこととする．

最後に対応するセルに処理を書き込むが，その前に状態変数に対する代入文をもつ処理を括弧 (<<<>>>) で括る．このようにして，遷移を意味する処理であることをわかりやすくする．現在は単純な代入文のみを状態変数の更新とみなしており，その他の方法による値の更新処理を括弧で括る方法は，現在検討中である．その後，対応するセルに処理文を書き込む．もし対応する処理がなければ，N/A とセルに表示する．本稿においては，最終的に表は tsv ファイルで作成され，着色やセル結合等の可読性を向上する機能を持っていない．以上によって状態遷移表を完成することができる．

3.3　実行結果

状態遷移を含む，あるソースコードをツールに入力した．イベント処理表と状態遷移表は，ツールによって TSV(Tab Separated Value) 形式で出力される．それぞれの TSV ファイルを表計算ソフトによって開き，手作業によって着色，罫線描画，セル結合などの加工を行ったものを図 7 および図 8 にそれぞれ示す．

この結果について考察する．まず，図 7 についてみると，イベントにすべての条件式の組み合わせが網羅されている．また，それに対して対応する処理が表示されている．従来手法では，これらのイベントに対応する処理を一意に定められない場合があったが，この表をみると，すべて一意なイベントであることがわかる．

次に，状態遷移表について考察する．状態遷移表では，今回状態変数の定義を満たしている変数 led_state を状態変数として選択した．図 8 においては，遷移を “<<<>>>” によって括っているため，状態の遷移がどのようなときに行われているかがわかりやすくなっている．また，イベントと状態の組み合わせが一意になっていることがわかる．

	A	B
1	Target :void Change_Ledcolor(void)	
2	swflg ==(1), led_state==0, dice_value2 == 1	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 1
3	swflg ==(1), led_state==0, dice_value2 == 2	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 2
4	swflg ==(1), led_state==0, !(dice_value2 == 2) && !(dice_value2 == 1)	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO COLOR CHANGE¥n")
5	swflg ==(1), led_state==1, dice_value2 == 1	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 2
6	swflg ==(1), led_state==1, dice_value2 == 2	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 0
7	swflg ==(1), led_state==1, !(dice_value2 == 2) && !(dice_value2 == 1)	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO COLOR CHANGE¥n")
8	swflg ==(1), led_state==2, dice_value2 == 1	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 0
9	swflg ==(1), led_state==2, dice_value2 == 2	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); led state = 1
10	swflg ==(1), led_state==2, !(dice_value2 == 2) && !(dice_value2 == 1)	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO COLOR CHANGE¥n")
11	swflg ==(1), !(led_state==2) && !(led_state==1) && !(led_state==0)	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO COLOR¥n")

図 7　イベント処理表 (セルの着色や罫線の加工を手作業で実施した．)

これらによって，この例においては**【問題点 1】**と**【問題点 2】**を解決することができたと考える．**【問題点 3】**については現在，問題点を解決できていないが，記述の通り割込みハンドラに対する実装方法を検討したため，実際に実装し効果を確認する必要がある．

4. 形式手法にもとづく分析支援の検討

4.1 Event-B の利用

ここまでの状態遷移表から，あるイベントとある状態の組み合わせに対する処理を表現することができている．これを使用し，状態遷移表の形式手法 Event-B [5] による定理証明とモデル検査を検討している．Event-B のモデルは，(グローバル) 変数とその不変条件，およびイベントから構成される．Event-B におけるイベントは，ガード条件とそのガード条件が成り立ったときに実行される処理から構成され，必要であればイベントへの引数や内部変数の定義をすることもできる．モデル中の処理に対し，代入文のみ記述することが許されている．

Event-B におけるイベントは，状態遷移表のある状態とイベントの組み合わせおよび，処理中の if-else 構文の数だけ定義される．また，Event-B モデルの各イベントにおけるガード条件は対応する状態遷移表の状態，イベントおよび処理中の条件式からなる．状態遷移表中の処理は，Event-B における対応するイベントの処理として定義される．加えて，ソースコードにおけるグローバル変数や入出力変数をモデルの変数として定義し，その変数の型が不変条件として定義される．

しかし，C 言語と Event-B の記法の間には差がある．この差を埋めるために，以下を制約として想定している．

- 条件式は，単純な比較文のみで構成される．
- プログラム中には定義されていない関数，定数，変数が存在しない．
- 標準ライブラリの関数は使用されない．

モデル検査においては，Event-B の開発環境である RODIN に ProB のプラグインを導入することによって，不変条件を満たしていることを検証できる．また，全てのイベントのガード条件が満たされておらず，イベントが何も起こり得ない場合にデッドロックを検出する．

4.2 NuSMV の利用

ここまでの状態遷移表から，あるイベントとある状態の組み合わせに対する処理を表現することができている．これを使用し，状態遷移表のモデル検査のためにモデル検査ツールである NuSMV [6] の利用を検討している．このとき，モデル検査を実施する目的はコード分析が目的であり，コードの検証を目的とはしていない．本稿において示した状態遷移表はいずれも規模の小さいものであるが，本来条件シーケンスはその組み合わせの数が非常に大きいことが予想される．そのため，抽出される状態遷移表の規模は非常に大きいことがある．そこで，状態遷移について視覚的に俯瞰できるだけではなく，コード理解のための分析支援があることが望ましい．たとえば，状態 S_1 から状態 S_2 への遷移は存在する/存在しないといったことを，容易に分析できるような機能を導入することで，ユーザの支援を行うことを考えている．

SMV のモデルは，変数定義部，遷移定義部，および仕様定義部からなっている．変数定義は各変数の宣言文より抽出され，また値域については宣言文およびコードにおける変数の使途から抽出することができる．次に，遷移定義部に関しては，2 通りの書き方があるが，今回，非決定的な記述は少ないと予想されるため，INIT および ASSIGN を用いて，初期値と次状態を定義することで状態遷移を記述することが可能だと考えている．INIT においては初期値を記述するが，初期値の取得は，そもそも初期化が存在しないなど，非決定的になることが予想される．しかし，ASSIGN においては，イベント処理表の処理部に記述されているある処理がイベント部の条件がすべて満たされたときに実行されると解釈でき，次状態の定義は容易であると考えられる．

最後に，時相論理式については，イベント処理表からは抽出しない．本手法ではユーザが記述するものとする．

しかし，C 言語と SMV の記法の間にも Event-B と同様に差がある．この差を埋めるために，Event-B と同様の制約を設定する．以上のように，状態遷移表から SMV におけるモデルを抽出し，コードの理解支援を実施する方法について検討した．

4.3 Event-B と SMV の比較検討

Event-B と SMV を実際に利用する状況を想定する．Event-B では，不変条件を利用してどんなときでもある性質を満たしていることを確認できることが強みであり，対象の検証のために使用することが有効であると考えられる．SMV では，時相論理式を使用することができるため，状態変数の遷移を時相論理式によって定義することで，ある遷移が存在するかしないかという検証に向いている．

すなわち，状態遷移を表現することについて SMV が優れているため，状態遷移表と供してコード分析支援のために使用さ

	A	B	C	D	E
1	function:void Change_Ledcolor(void)	target:led_state			
2		0	1	2	!(led_state==2)&&!(led_state==1)&&!(led_state==0)
3	swflg ==(1), dice_value2 == 1	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 1;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 2;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 0;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR¥n");
4	swflg ==(1), dice_value2 == 2	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 2;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 0;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); <<<led_state = 1;>>>	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR¥n");
5	swflg ==(1), !(dice_value2 == 2) && !(dice_value2 == 1)	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR_CHANGE¥n");	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR_CHANGE¥n");	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR_CHANGE¥n");	int dice_value1, dice_value2; dice_value1 = Roll_Dice(); printf("Switch_ON¥n"); printf("NO_COLOR¥n");
6	!(swflg ==(1))	N/A	N/A	N/A	N/A

図 8　状態遷移表 (セルおよび文字の着色，罫線およびセル結合の加工を手作業で実施した．)

れることに適している．しかしながら，Event-B も強力なツールである．コード分析支援を行うために使用するのではなく，C 言語からリバースエンジニアリングによって作成されたモデルを検証することには意味がある．

しかしながら，C 言語特有の表現であるポインタなどに対してどのように対処するかは今後の課題となる．加えて，どちらを使用しても状態爆発の可能性があり，大規模なシステムへの適用は困難となるが，今後，小規模なシステムに対して形式手法 SMV によるコード分析支援を実装していく考えである．

5. 関連研究

組込みシステムがネットワークに接続されることが一般化され，組込みシステムの複雑化・大規模化が問題となってきている [7], [8]．複数の組込み機器を通信ネットワークで接続したサービスを実現するためには，それぞれの組込み機器が持つ膨大な数の状態と様々な処理を理解しながら開発を行う必要がある [8]．このような複雑なサービスの開発において，状態遷移表が用いられている [8], [9]．

状態遷移図の自動抽出を目的とした手法がいくつか提案されているが，そのほとんどは実行時情報をもとに生成する手法である [10]〜[12]．本研究で提案している手法はレガシーコードを対象としており，レガシーコードにはテストコードやテストケースなどテストに関する成果物や文書が古いもしくは欠如していることがほとんどである．そのため，本研究で対象としているレガシーコードに対して，実行時情報が必要とする手法を適用することは現実的ではない．Walkinshaw らは，抽象実行を用いてソースコードから状態遷移図を抽出する手法を提案している [13]．彼らの手法が抽出する状態遷移図は，メソッド呼出し文や例外処理の発生による遷移のみを扱う粒度の大きいものである．本研究は，状態遷移図ではなく状態遷移表を抽出する．また，条件分岐による遷移を扱っており，抽出する状態遷移の粒度が異なる．

6. おわりに

本研究においては，従来手法における問題点を解決するべく，従来手法を改良する形でソースコードから状態遷移表を抽出する手法を提案した．また，手法をツールとして実装した結果，JIS 規格を意識し，状態とイベントの網羅性を備えた状態遷移表が抽出された．最後に，形式手法を用いたコード分析支援について検討した．

今後の課題は，実際に形式手法を用いてコード分析支援につなげるためのツール化を検討することである．

謝辞　ツールの開発や実験にご協力いただいた名古屋大学大学院情報科学研究科 若林 丈紘氏およびツール対するご意見を賜りました，組込みシステム技術協会 状態遷移設計研究 WG の皆様に感謝致します．

文　献

[1] 山本椋太，吉田則裕，竹田彰彦，舘伸幸，高田広章，"組込みソフトウェアを対象とした状態遷移表抽出手法，" 電子情報通信学会技術研究報告，vol.116，no.127，pp.13–18，2016．
[2] 山本椋太，"組込みソフトウェアを対象とした状態遷移表抽出手法とそのモデル検査への応用について，" ウィンターワークショップ 2017・イン・飛騨高山，vol.2017，pp.53–54，2017．
[3] JIS, "X 0131–1995, ソフトウェアの状態遷移の構成及びその表記方法，" 1995．
[4] 渡辺政彦，拡張階層化状態遷移表設計手法 Ver. 2. 0: EmbeddedSE のための設計手法，キャッツ，1998．
[5] J.-R. Abrial, Modeling in Event-B: system and software engineering, Cambridge University Press, 2010.
[6] K.L. McMillan, "Model checking," PhD thesis, The school of the computer Science, Carnegie Mellon University, 2003.
[7] 高田広章，"組込みシステム開発技術の現状と展望，" 情報処理学会論文誌，vol.42，no.4，pp.930–938，2001．
[8] 渡辺政彦，"状態遷移ベースのソフトウェア開発環境の現状と動向，" 計測と制御，vol.41，no.2，pp.117–121，2002．
[9] 松崎寛之，"レガシーデバイスを用いたホームネットワーク構築における機器の状態取得及び管理に関する研究，" Master's thesis，北陸先端科学技術大学院大学 情報科学研究科，2005．
[10] A.W. Biermann and J.A. Feldman, "On the synthesis of finite-state machines from samples of their behavior," IEEE Trans. Comput., vol.C-21, no.6, pp.592–597, 1972.
[11] D. Lorenzoli, L. Mariani, and M. Pezzè, "Inferring state-based behavior models," Proceedings of the 2006 International Workshop on Dynamic Systems Analysis, pp.25–32, 2006.
[12] T. Xie and D. Notkin, "Automatic extraction of object-oriented observer abstractions from unit-test executions," Proceedings of International Conference on Formal Engineering Methods, pp.290–305, 2004.
[13] N. Walkinshaw, K. Bogdanov, S. Ali, and M. Holcombe, "Automated discovery of state transitions and their functions in source code," Software Testing, Verification & Reliability, vol.18, no.2, pp.99–121, 2008.