「状態遷移表のリバースモデリングへの適用」

2013年11月21日

状態遷移設計研究会

キャッツ株式会社　竹田彰彦

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

現在の組込みソフト開発の状況

■ソフト開発の傾向

- 短い期間で開発(改修)を終了させて、納品しなければならない。
- 当然のことながら、不具合を発生させてはならない。
- 対象とする製品のコードサイズが大きくなってきている。
- 長年利用しているコードを対象にするので、追加/改修が複数回行われている。
- 改修を担当する技術者（会社）も違うこともある。
- ドキュメントは初期バージョンのものはあるが、ドキュメント修正時間が確保できないのでメンテナンスできない。ソースコードが最新の仕様書(!)ということも。

ソースコードのブラックボックス化が進行中！

開発／改修を強行すると。。。

- □ 改修内容について確認しようとしても、正しいかどうか判断できない可能性
- □ 改修時に不具合を作り込む可能性
- □ 改修に問題はなくても、対象外の機能に影響を与える可能性
- □ 改修の結果、潜在していた不具合が発現する可能性

新たな取り組みに移行できない背景

オブジェクト指向設計、モデルベース開発、コンポーネント開発、ソフトウエアプロダクトライン開発など効率化を謳った手法は、様々ある。
しかしながら、開発プロセス、設計手法の移行は進まない。

レガシーコードの存在が足かせになっている！

派生開発が主流の開発で、既存コードの設計を踏襲せざるを得ない。

■リバースエンジニアリング

そこで今、注目を集めているのが「リバース・エンジニアリング」
リバース・エンジニアリングとは、現行のシステム（ソースコードやデータベース）を解析し、その仕様を明らかにする技術。

組込みシステムで状態遷移設計は普遍的なモデル手法。

レガシーコードの中には、状態遷移が必ずあるはず。

それを、見つけられれば・・・

■【仮説】

「フラグがある所に、状態はある。!!」

■ 2013年度のテーマ

「状態遷移表のリバースモデリングへの適用」

⇒レガシーコードから、状態を抽出し状態遷移表モデルを導出する手法の研究。

一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

■　状態変数の定義

- 状態変数は有限個である。
- 状態変数は、内部で更新される。

■　イベント変数の定義

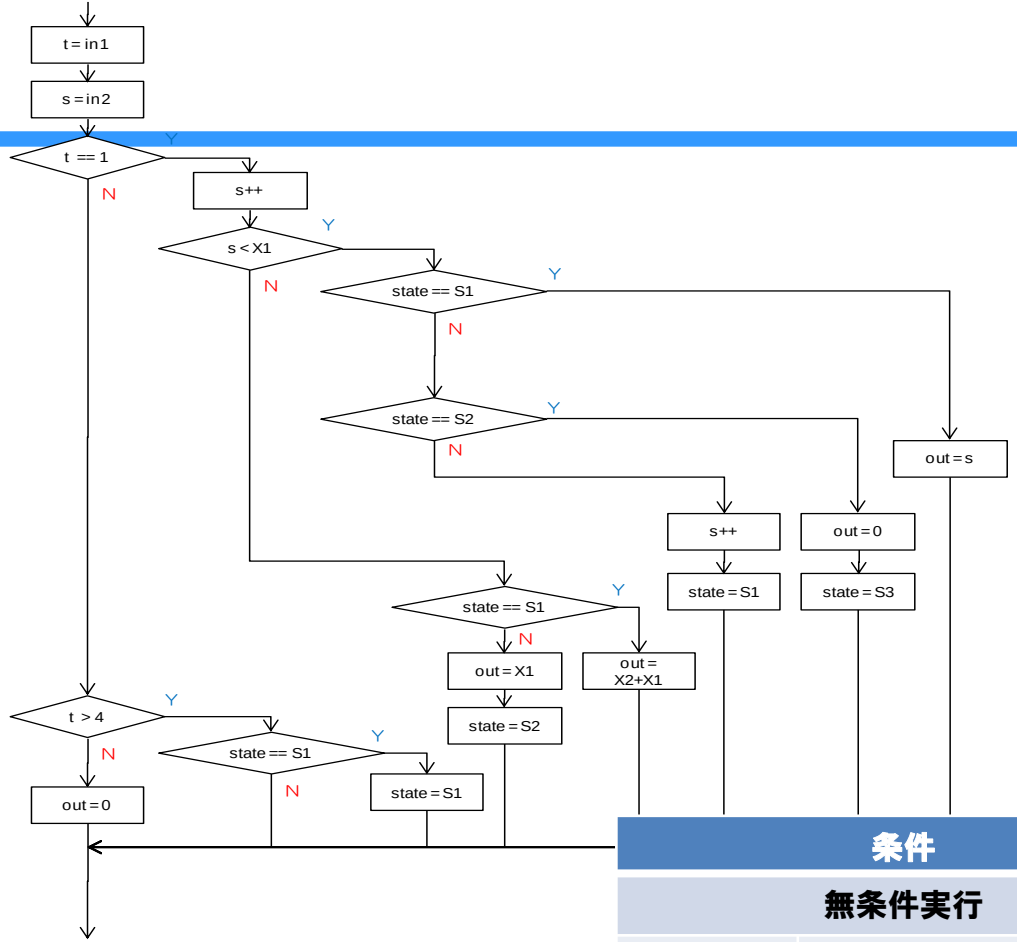
- イベント変数は、外部から渡される。
- 状態変数を更新するアクションの対応判定変数は
  イベント変数である。
- イベント変数は内部では更新されない

リバースモデリングの原理

```
extern int in1,in2
int state;
#define S1  1
#define S2  2
#define S2  3
#define X1 5
#define X2 T1*2
int out

void task(void)
{
   int t,x;

   t = in1;
   s = in2;
   if ( t == 1) {
      s++;
      if (s < X1) {
         if(  state == S1 ){
            out = s;
         else if( state == S2 ){
            out = 0;
            state = S3
         else{
             s++;
            state = S1;
         }
      } else {
         if(  state == S1 ){
            out = X2+X1;
         else{
            out = X1;
            state = S2
      }
   }
   else  if(  t > 4 ){
        if(  state == S1 ){
            out = s;
        }
   }
   else{
       out = 0;
    }
}
```



条件分岐をif－else構造でモデル化
For、whileのloop構造は処理とする
SW、caseもif－else構造にモデル置換

条件のない処理は無条件実行イベント

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	（Out=S）
		State==S2	（Out=0),State=S3
		else	S++,State＝S1
	else	State==S1	（Out=X2+X1）
		else	（Out=X1),State=S2
T>4	Sate==S1		（Out=S）
else			（Out=0）

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	（Out=S）
		State==S2	（Out=0),State=S3
		else	S++,State＝S1
	else	State==S1	（Out=X2+X2）
		else	（Out=X1),State=S2
T>4	Sate==S1		（Out=S）
else			（Out=0）

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	（Out=S）
		State==S2	（Out=0),State=S3
		else	S++,State＝S1
	else	State==S1	（Out=X2+X1）
		else	（Out=X1),State=S2
T>4	Sate==S1		（Out=S）
	else		/
else			（Out=0）

条件分岐をif－else構造でモデル化

強制的にelseを追加

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		else	S++,State＝S1
	else	State==S1	(Out=X2+X1)
		else	(Out=X1),State=S2
T>4	Sate==S1		(Out=S)
	else		/
else			(Out=0)

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		State==S3	S++,State＝S1
	else	State==S1	(Out=X2+X1)
		State==S2	(Out=X1),State=S2
		State==S3	(Out=X1),State=S2
T>4	Sate==S1		(Out=S)
	State==S2		/
	State==S3		/
else			(Out=0)

StateはS1,S2,S3の有限値
Else条件を置換

一般社団法人 組込みシステム Japan Embedded Systems Technology Association

条件			State		
			S1	S2	S3
無条件実行			T=in1,S=in2	T=in1,S=in2	T=in1,S=in2
T==1			S++	S++	S++
	S<X1		(Out=S)	/	/
			/	(Out=0),State=S3	/
			/	/	S++,State=S1
	else		(Out=X2+X1)	/	/
			/	(Out=X1),State=S2	/
			/	/	(Out=X1),State=S2
T>4			(Out=S)	/	/
			/	/	/
			/	/	/
else			(Out=0)	(Out=0)	(Out=0)

Stateは有限個の変数
内部更新がある
Stateを状態変数として定義

Sは状態の更新があるので
イベントと等価の条件である
Tは、単なる条件である

		State		
		S1	S2	S3
無条件実行		T=in1,S=in2	T=in1,S=in2	T=in1,S=in2
T==1		S++	S++	S++
	S<X1	(Out=S)	(Out=0),State=S3	S++,State=S1
	else	(Out=X2+X1)	(Out=X1),State=S2	(Out=X1),State=S2
T>4		(Out=S)	/	/
else		(Out=0)	(Out=0)	(Out=0)

組込みシステム
Japan Embedded Systems Te

		State		
		S1	S2	S3
無条件実行		T=in1,S=in2	T=in1,S=in2	T=in1,S=in2
T==1		S++	S++	S++
	S<X1	(Out=S)	(Out=0),State=S3	S++,State＝S1
	else	(Out=X2+X1)	(Out=X1),State=S2	(Out=X1),State=S2
T>4		(Out=S)	/	/
else		(Out=0)	(Out=0)	(Out=0)

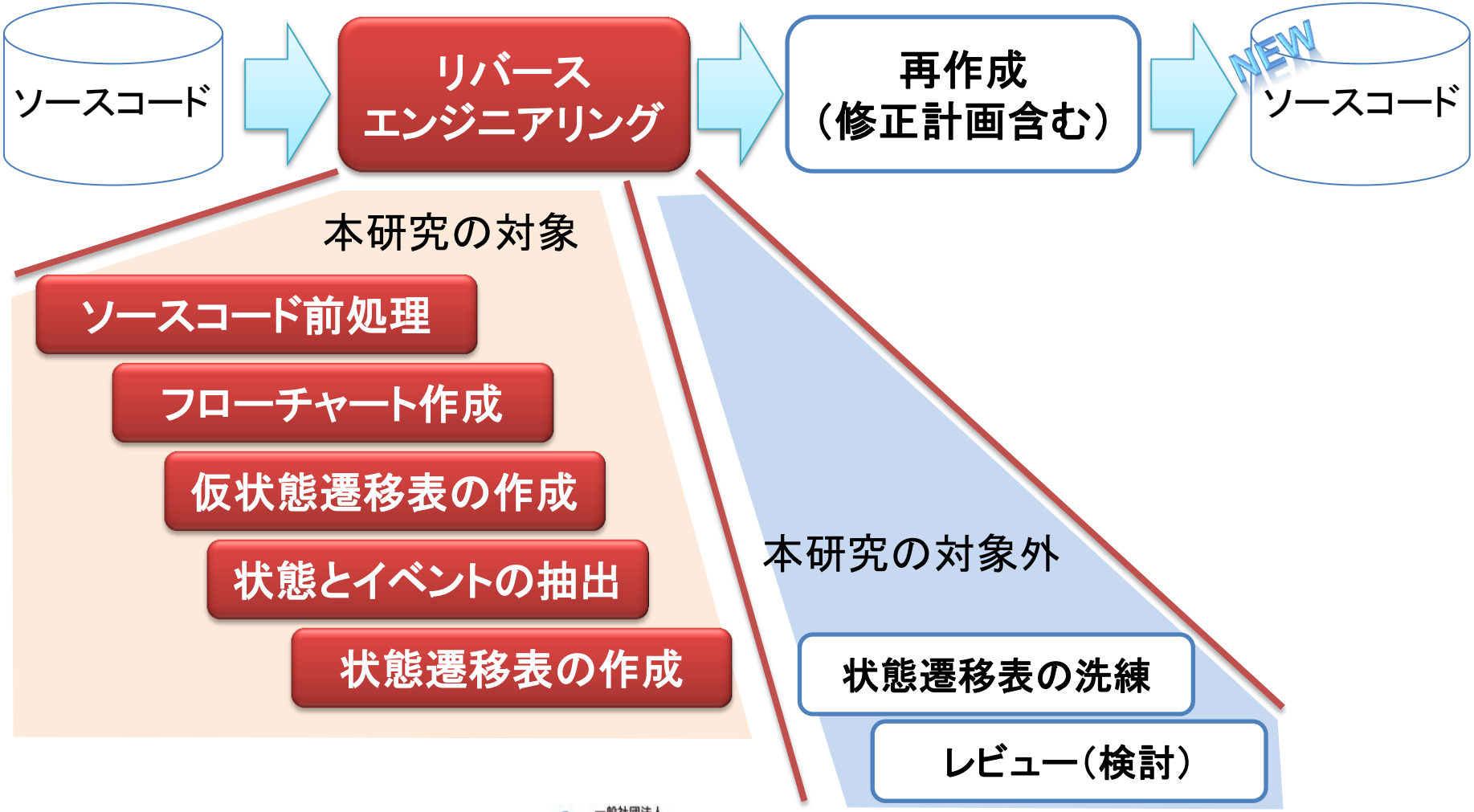
状態遷移表モデルの範囲

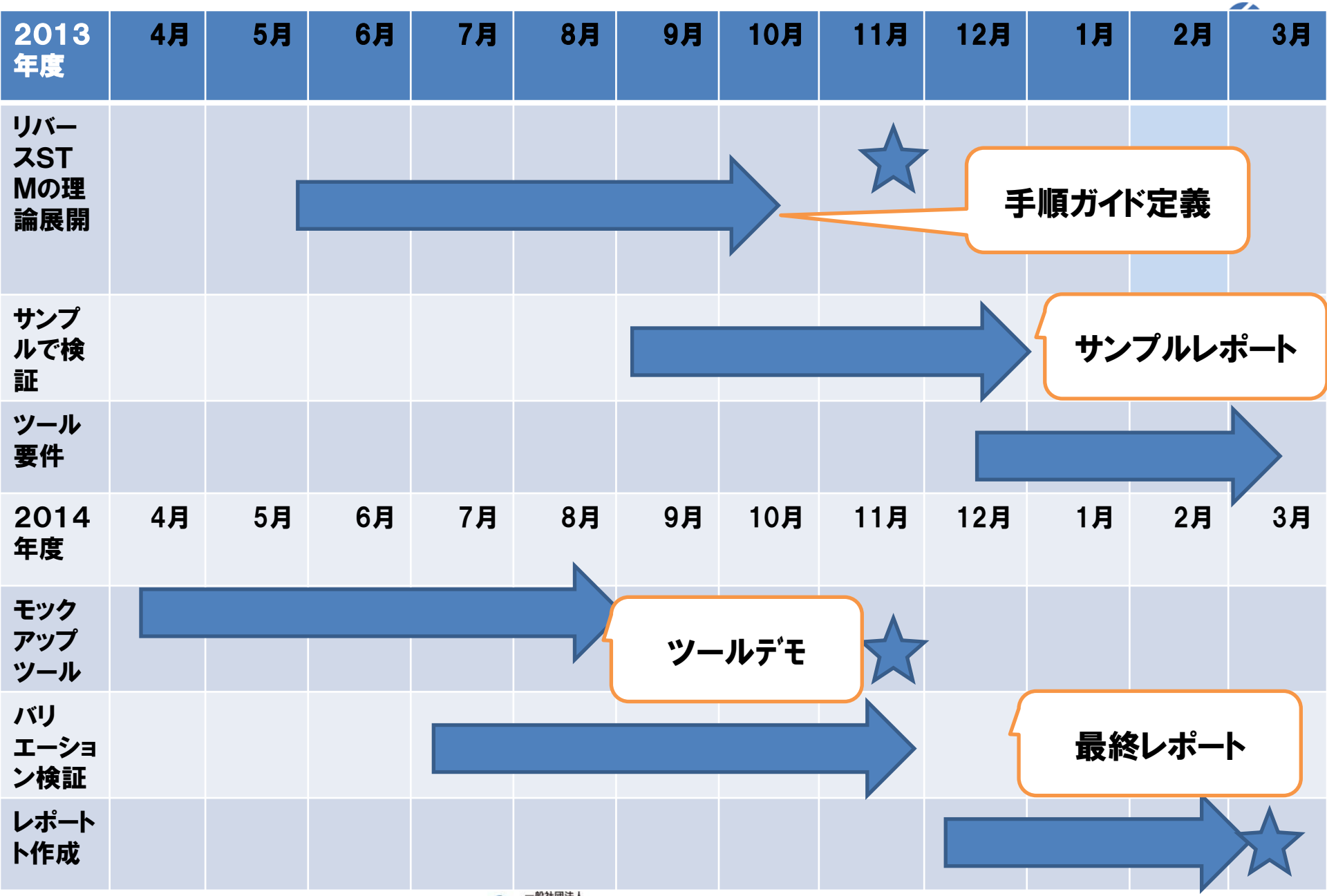
	State		
	S1	S2	S3
S<X1	(Out=S)	(Out=0), State=S3	S++, State＝S1
else	(Out=X2+X1)	(Out=X1), State=S2	(Out=X1) ,State=S2

一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

本研究の対象範囲

■【リファクタリングの流れと研究対象範囲】

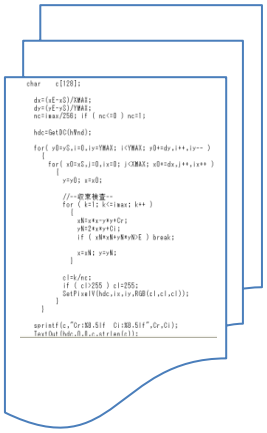




一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

リバースモデリング作業手順

リバースモデリングの作業手順



①レガシーコードの整形（ツール化検討）

・コメント、＃ifdefを削除する。
・解析範囲のtop関数を指定する。？

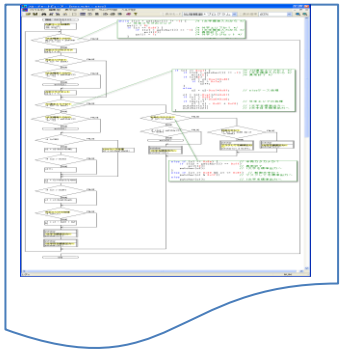
②スケルトンコードの生成（ツール化検討）

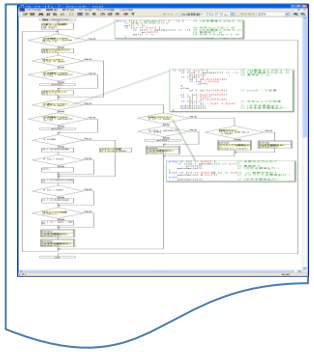
分岐条件と変数の更新位置をgrep。？

リバースモデリング＆ZIPCマイグレーションの作業手順

③構文ツリーの作成（解析・構図）

フローチャートを生成し、
構文ツリー図を作成する





④状態遷移表の作成

分岐条件を階層化した状態遷移表を
作成する

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		else	S++,State＝S1
	else	State==S1	(Out=X2+X2)
		else	(Out=X1),State=S2
T>4	Sate==S1		(Out=S)
else			(Out=0)

リバースモデリングの作業手順(④補足)

仮状態遷移表の作成

条件分岐のインデントと仮状態遷移表の条件階層は一致している

ARC_main()			
if (ARC_REG.STATUS.BIT.RI == 1)	ARC_ReadData(); header = (struct ARC_HEADER_FORMAT*)&ARC_RxData[ARC_RxData[2]]; ARC_JyohenSendReq.NodeID[ARC_RxData[0]] = 1; ARC_JyohenGpsSendReq.NodeID[ARC_RxData[0]] = 1;		
		if((header->CMND == 0x01)&& (header->INT == 0x00)&& (FLG.remotelocal == FLGREMOTE))	AntContLog(ANTLOG_INFO, "[REMOTE]"); ARC_CtrlReq((struct CONTROL_DEMAND*)header); LogRTC();
	if (header->LINE == 0xf7 && header->ATB == 0x01)	if((header->CMND == 0x10)&& (header->INT == 0x00))	ARC_StatusSendReq.Req = 1; ARC_StatusSendReq.NodeID = header->SID;
		if((header->CMND == 0x10)&& (header->INT == 0x01))	ARC_StatusGpsSendReq.Req = 1 ARC_StatusGpsSendReq.NodeID = header->SID;

リバースモデリングの作業手順

仮状態遷移表の作成

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		else	S++,State=S1
	else	State==S1	(Out=X2+X2)
		else	(Out=X1),State=S2
T>4	Sate==S1		(Out=S)
else			(Out=0)

⑤不足しているelseを追加する

・ソースコード上で省略されている
「else」を追加する

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		else	S++,State=S1
	else	State==S1	(Out=X2+X1)
		else	(Out=X1),State=S2
T>4	State==S1		(Out=S)
	else		/
else			(Out=0)

多くの場合、処理の必要がないときには、
「else」「case」「default」が省略されている。

「処理がない」場合、状態遷移表の
処理記載部分には「／」で表現する。

リバースモデリングの作業手順

状態とイベントの抽出

状態遷移表の作成

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		else	S++,State=S1
	else	State==S1	(Out=X2+X1)
		else	(Out=X1),State=S2
T>4	State==S1		(Out=S)
	else		/
else			(Out=0)

⑥elseを有限値に展開する

•「else」をとりうる値に展開する
（手順⑤で入れた else を値に変える）

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		State==S3	S++,State=S1
	else	State==S1	(Out=X2+X1)
		State==S2	(Out=X1),State=S2
		State==S3	(Out=X1),State=S2
T>4	State==S1		(Out=S)
	State==S2		/
	State==S3		/
else			(Out=0)

条件			処理
無条件実行			T=in1,S=in2
T==1			S++
	S<X1	State==S1	(Out=S)
		State==S2	(Out=0),State=S3
		State==S3	S++,State=S1
	else	State==S1	(Out=X2+X1)
		State==S2	(Out=X1),State=S2
		State==S3	(Out=X1),State=S2
T>4	State==S1		(Out=S)
	State==S2		/
	State==S3		/
else			(Out=0)

⑦状態とイベントの抽出

•状態、イベントを抽出し、
状態遷移表を編集する

		State		
		S1	S2	S3
無条件実行		T=in1,S=in2	T=in1,S=in2	T=in1,S=in2
T==1		S++	S++	S++
	S<X1	(Out=S)	(Out=0),State=S3	S++,State=S1
	else	(Out=X2+X1)	(Out=X1),State=S2	(Out=X1),State=S2
T>4		(Out=S)	/	/
else		(Out=0)	(Out=0)	(Out=0)

一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

リバースモデリングの作業手順（本研究対象外）

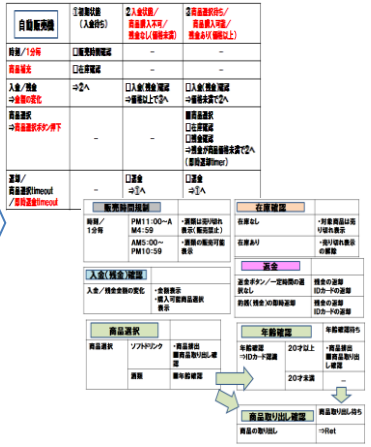
状態遷移表の洗練

レビュー（検討）

		State		
		S1	S2	S3
無条件実行		T=in1.S=in2	T=in1.S=in2	T=in1.S=in2
T==1		S++	S++	S++
	S<X1	(Out=S)	(Out=0),State=S3	S++,State=S1
	else	(Out=X2+X1)	(Out=X1),State=S2	(Out=X1),State=S2
T>4		(Out=S)	/	/
else		(Out=0)	(Out=0)	(Out=0)

⑧状態遷移表の最適化

- 状態遷移表の最適化を検討する
  - −状態／イベントの抽象化
  - −状態遷移表の階層化



サンプル検証

スロットマシーンを作成せよ！

要求仕様

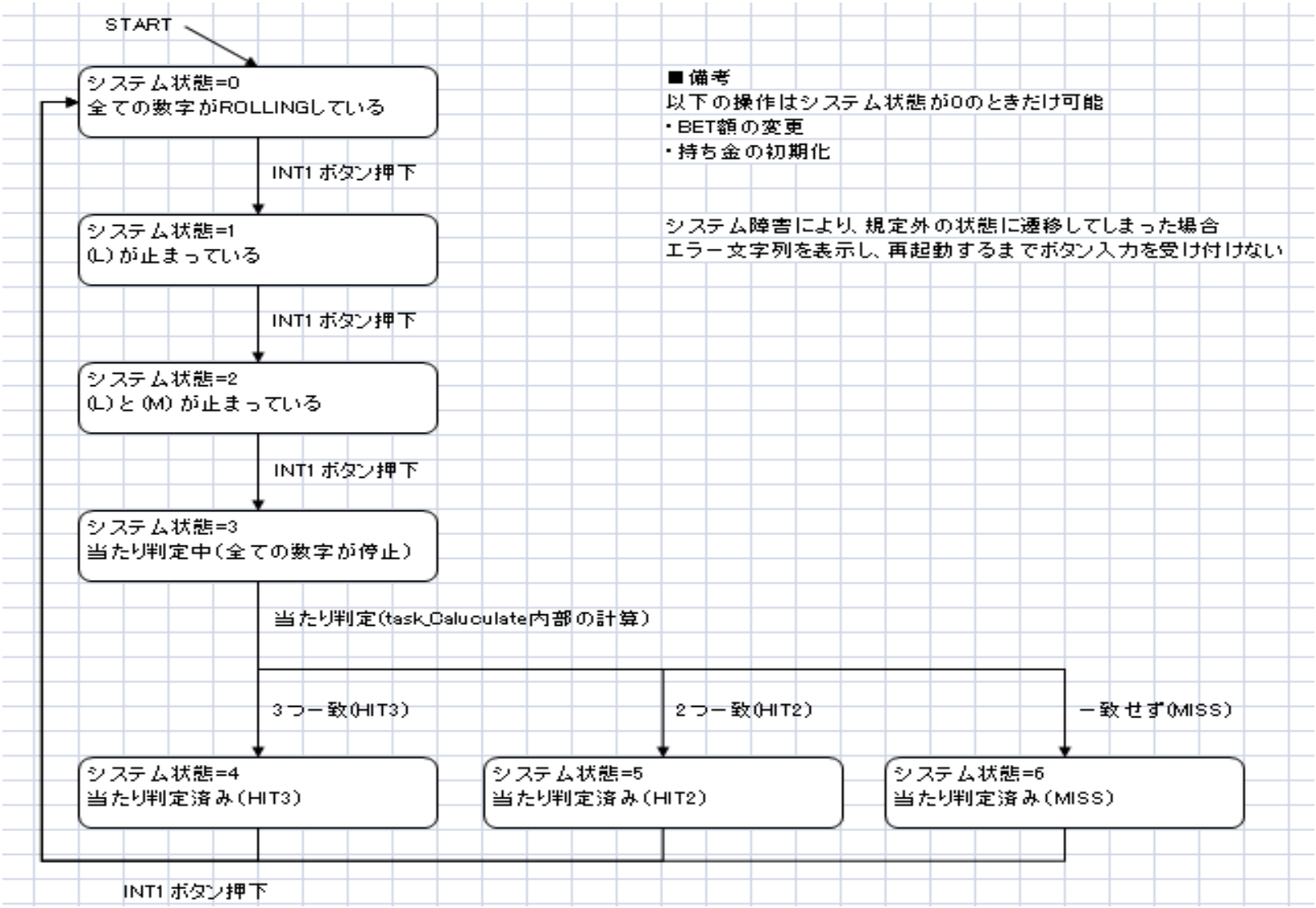
・INTボタン押下で、3つのスロットを左から順に止め、
その結果により、持ち金の増減を行う。

・起動直後は、すべてrollingさせる。
・すべてrollingの状態の時、掛け金を設定できる。
・1回目のINT押下で、掛け金を確定し、左のスロットを止める。
・2回目のINT押下で、真ん中のスロットを止める。
・3回目のINT押下で、右のスロットを止めると同時に、
止まった3つの絵柄を比較し、その結果で持ち金を増減させる。

・3つとも同じ絵柄のとき、掛け金の5倍を持ち金に加える。
・2つが同じ絵柄のとき、掛け金を返す。
・3つとも異なる絵柄のとき、掛け金を没収する。

・4回目のINT押下で、すべてrolling状態とする

状態遷移図サンプル



サンプルソースコード

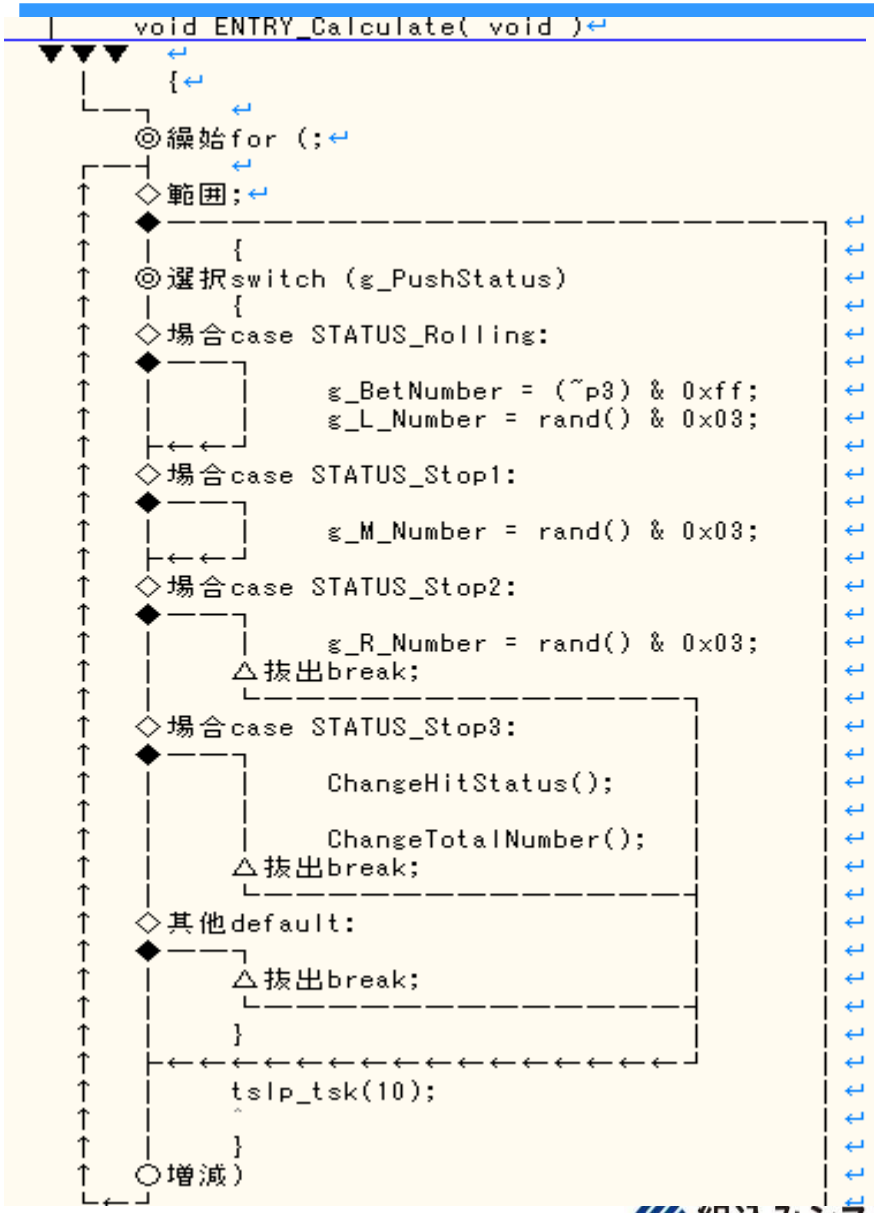
```
void ENTRY_Calculate( void )
{
        for (;;)
        {
                switch (g_PushStatus){
                        /* まだ ROLLING 中の数字がある場合 */
                        case STATUS_Rolling:
                                g_BetNumber = (~p3) & 0xff; /*
        BET 額を更新 */
                                  g_L_Number = rand() & 0x03; /* (L) を乱数で更
        新 下二桁 bit(0-3) を取る */
                        case STATUS_Stop1:
                                g_M_Number = rand() & 0x03;
        /* (M) を乱数で更新 下二桁 bit(0-3) を取る */
                        case STATUS_Stop2:
                                g_R_Number = rand() & 0x03; /*
        (R) を乱数で更新 下二桁 bit(0-3) を取る */
                        break;
                        /* 全ての数字が止まった場合（当たり判定） */
                        case STATUS_Stop3:
                                ChangeHitStatus();   /* 当たり
        判定を行い、ステータスを遷移 */
                                ChangeTotalNumber(); /* ステ
        ータスに従い、持ち金を更新 */
                                break;
                        default:
                                break;
                }
                tslp_tsk(10);        /* 10( * 10msec) スリープする */
        }
}
```

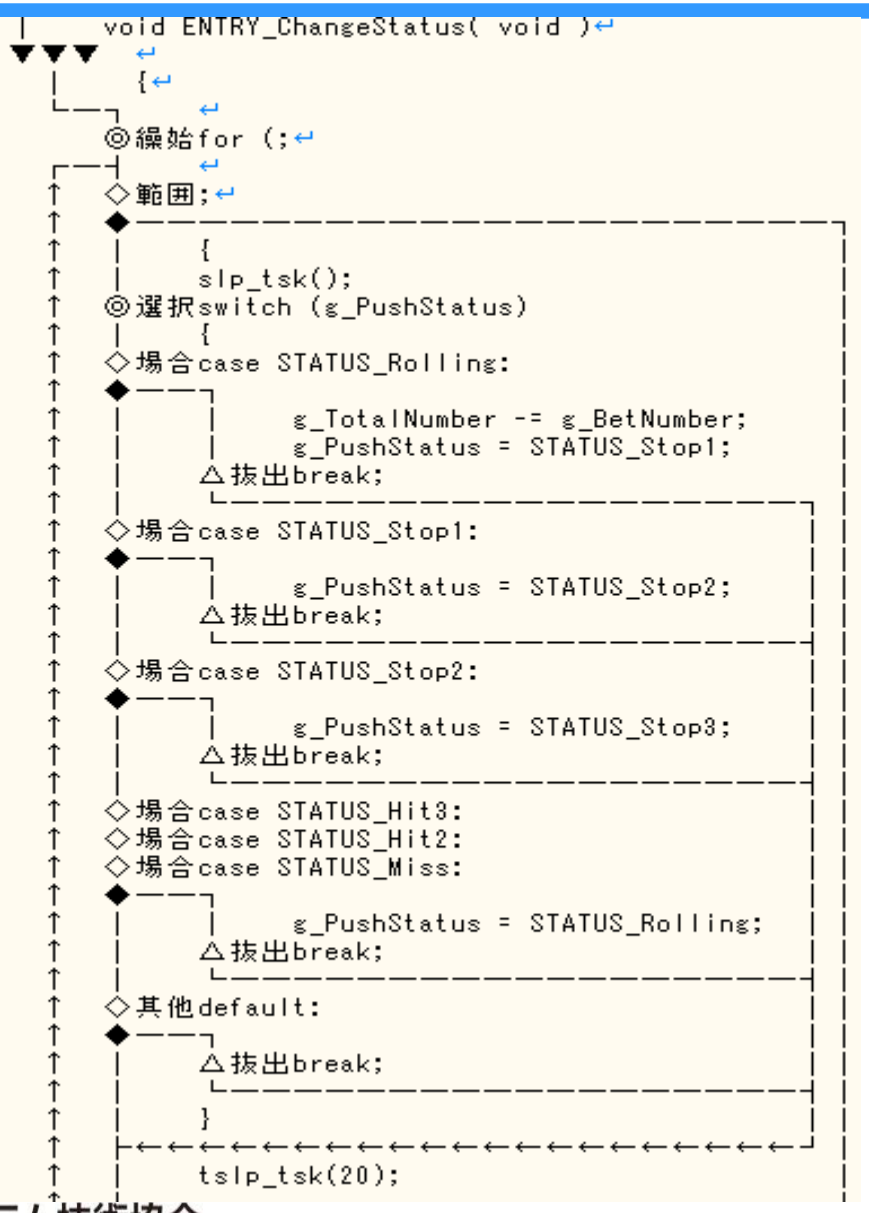
```
void ENTRY_ChangeStatus( void )
{
        for (;;)
        {
                slp_tsk(); /* INT1 ボタン待ち */

                /* システム状態を遷移 */
                switch (g_PushStatus){
                        /* まだ ROLLING 中の数字がある場合は次の状態へ */
                        case STATUS_Rolling:
                                g_TotalNumber -= g_BetNumber; /*
        Bet */
                                g_PushStatus = STATUS_Stop1;
                        break;
                        case STATUS_Stop1:
                                g_PushStatus = STATUS_Stop2;
                        break;
                        case STATUS_Stop2:
                                g_PushStatus = STATUS_Stop3;
                        break;
                        /* 当たり判定済みの場合は ROLLING (初期状態)へ */
                        case STATUS_Hit3:
                        case STATUS_Hit2:
                        case STATUS_Miss:
                                g_PushStatus = STATUS_Rolling;
                        break;
                        /* それ以外の状態 */
                        default:
                                break;
                }

                tslp_tsk(20);     /* チャタリング防止 */
                int1ic = INT1_OK; /* INT1 割込み許可（ハンドラ起動時に禁止） */
        }
}
```

サンプルフローチャート





組込みシステム技術協会
Japan Embedded Systems Technology Association

サンプルSTM　・・・ sample1

関数	条件	g_PushStatus						
		Rolling	Stop1	Stop2	Stop3	Hit2	Hit3	Miss
ENTRY_Calculate	※((g_L_Number == g_M_Number) && (g_M_Number == g_R_Number))	/	/	/	g_PushStatus = STATUS_Hit3; g_TotalNumber += ((g_BetNumber << 2) + g_BetNumber	/	/	/
	※((g_L_Number == g_M_Number) || (g_L_Number == g_R_Number) || (g_M_Number == g_R_Number))	/	/	/	g_PushStatus = STATUS_Hit2; g_TotalNumber += g_BetNumber;	/	/	/
	※else	/	/	/	g_PushStatus = STATUS_Miss;	/	/	/
ENTRY_ChangeStatus		g_TotalNumber -= g_BetNumber; g_PushStatus = STATUS_Stop1;	g_PushStatus = STATUS_Stop2;	g_PushStatus = STATUS_Stop3;	/	g_PushStatus = STATUS_Rolling;	g_PushStatus = STATUS_Rolling;	g_PushStatus = STATUS_Rolling;

・rolling/stop1/stop2/stop3とHit2/Hit3/Missは状態の粒度が違う。
・stop3状態での内部条件。

サンプルSTM ・・・ sample2

関数	条件	flag_b1_c			
		0	1	2	3

Entry4	L,M,R 一致しない	int0ic=INT0_NG;
		money=money-bet;
		flag_b1_c=1;
			flag_b1_c=2;	MISS表示
				i=0;
				flag_b1_c=3;
					flag_b1_c=0;

	L,M,R 1つ一致	int0ic=INT0_NG;
		money=money-bet;
		flag_b1_c=1;
			flag_b1_c=2;	money=money+bet;
				HIT2 表示
				i=0;
				flag_b1_c=3;
					flag_b1_c=0;

	L,M,R 2つ以上一致	int0ic=INT0_NG;
		money=money-bet;
		flag_b1_c=1;
			flag_b1_c=2;	money=money+5*bet;
				HIT3 表示
				i=0;
				flag_b1_c=3;
					flag_b1_c=0;

・状態変数が意味不明、状態を0,1,2,3の直値で指定
・条件は状態2の時のみ有効

サンプルSTM　・・・ sample3

関数	条件			int1cnt 0	1	2	3

entry3	◇total >= bet			total -= bet;
				int0ic = INT0_NG;
				int1cnt++;
					×	×	×

	else	◇ranNum1 == ranNum2	◇ranNum1 == ranNum3	int1cnt++;	int1cnt++;	int1cnt++;
						total += bet * 5;
							int0ic = INT0_OK;
							int1cnt = 0;
							betFlag = ENABLE;

			else	int1cnt++;	int1cnt++;	int1cnt++;
						itotal += bet;
							int0ic = INT0_OK;
							int1cnt = 0;
							betFlag = ENABLE;

		◇(ranNum1 == ranNum3) || (ranNum2 == ranNum3)		int1cnt++;	int1cnt++;	int1cnt++;
						itotal += bet;
							int0ic = INT0_OK;
							int1cnt = 0;
							betFlag = ENABLE;

		else		int1cnt++;	int1cnt++;	int1cnt++;	int0ic = INT0_OK;
							int1cnt = 0;
							betFlag = ENABLE;

・状態を++で加算、状態の追加などメンテナンス困難
・条件は状態2の時のみ有効

サンプルSTM　・・・ sample4

ENTRY_NewLogicTask(void)		
無条件	PressCounter++;	
f(PressCounter==1)	bAmount=(int)p3;	
	if(bAmount>Amount)	bAmount=Amount;
	Amount-=(int)bAmount;	
f(PressCounter==3)	if(r1 ==r2 && r2==r3)	Amount += (bAmount*5);
	else if(r1 ==r2 || r1 ==r3 || r2==r3)	Amount += bAmount;
f(PressCounter==4)	if(Amount>0)	PressCounter=0;

・状態を++で加算、状態の追加など
　メンテナンス困難
・無条件に加算したのち、0に戻す。危険！

	PressCounter				
	0	1	2	3	4
	PressCounter++;	PressCounter++;	PressCounter++;	PressCounter++;	PressCounter++;

		bAmount=(int)p3;
		if(bAmount>Amount)のとき
		bAmount=Amount;
		Amount-=(int)bAmount;
				if(r1 ==r2 && r2==r3)のとき
				Amount += (bAmount*5);
				if(r1 ==r2 || r1 ==r3 || r2==r3)のとき
				Amount += bAmount;
					if(Amount>0) のとき
					PressCounter=0;

一般社団法人 組込みシステム技術協会 Japan Embedded Systems Technology Association

サンプルSTM　・・・ sample5

entry5(void)		
if (judge_int1 == OFF)	judge_int1 = ON; INT1_cnt++;	
	if (INT1_cnt == 1)	total -= bet;
	else if (INT1_cnt == 3)	iwup_tsk(&task4);
	else if (INT1_cnt >= 4)	INT1_cnt = 0; judge4 = OFF;
	judge_int1 = OFF;	

entry4(void)	
無条件	judge4 = ON;
if ((roll1 == roll2) && (roll2 == roll3))	total += bet * 5;
else if ((roll1 == roll2) || (roll2 == roll3) || (roll3 == roll1))	total += bet;
else	/

	INT1_cnt				
	0	1	2	3	4
	INT1_cnt++;	INT1_cnt++;	INT1_cnt++;	INT1_cnt++;	INT1_cnt++;
	/	total -= bet;	/	wup_tsk(&task4);	INT1_cnt = 0; judge4 = OFF;

- 状態を++で加算、状態の追加などメンテナンス困難
- 無条件に加算したのち、0に戻す。危険！
- タスクでベットの判定、タスクの必要はない。

適用効果の考察

1. **機械的にソースコードから状態遷移表を作成するプロセス**が見えてきた。
2. 振る舞いの可視化、モデル化により**レビューがしやすくなる**。
   振る舞いの漏れ・抜けが発見できる。
3. **状態変数名の変更**など、リファクタリングの要素を抽出できる。
4. 言語に依存せず、**すべてのコードに適用**できる。

状態遷移設計研究会では、このような活動に協力していただけるメンバーを募集しています。
ＪＡＳＡ技術本部にご連絡ください。
よろしくお願いします。

ご清聴ありがとうございました。

一般社団法人
組込みシステム技術協会
Japan Embedded Systems Technology Association

2013年度　状態遷移設計研究会活動紹介
「状態遷移表のリバースモデリングへの適用」

2013/11/21 発行

発行者　一般社団法人 組込みシステム技術協会
東京都中央区日本橋浜町1丁目8-1
TEL: 03(5821)7973 FAX: 03(5821)0444
URL: http://www.jasa.or.jp