一般社団法人　組込みシステム技術協会　　技術本部　　技術高度化委員会

状態遷移設計研究会

状態遷移表のリバースモデリングへの適用

状態遷移設計研究会の目的

状態遷移設計研究会は、状態遷移設計の漏れ・抜けに気づきやすい、という特性を持つ*「状態遷移表」*を広く普及、定着させることを目的に活動をしています。
これまでの研究成果は、
①状態遷移表の標準的な記法の定義
②ソフトウエアプロダクトライン開発（SPLE）と状態遷移表設計の融合プロセスのガイド
です。
この融合プロセスは、再利用性の高い状態遷移表を作成する手順を解説したもので、*「状態遷移表設計におけるSPLE実践ガイド」*として公開しました。
そして、本年度より*「状態遷移表のリバースモデリングへの適用」*という新たなテーマで活動しています。

背景と仮説

■ソフトウエア開発の課題と傾向
・システム障害やトラブル対応に戸惑う。
・システムの改修に膨大な調査費用がかかる。
・ドキュメントがなく、ソースコードだけが頼りで、解析できない。
・関係者が退職し、拠り所もない。

⇨ ソースコードのブラックボックス化が進行中！

新たな取り組みに移行できない背景

⇨ レガシーコードの存在が足かせになっている！

■解決策
・リバースモデリングで、既存システムをソースコードからモデル化。

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。

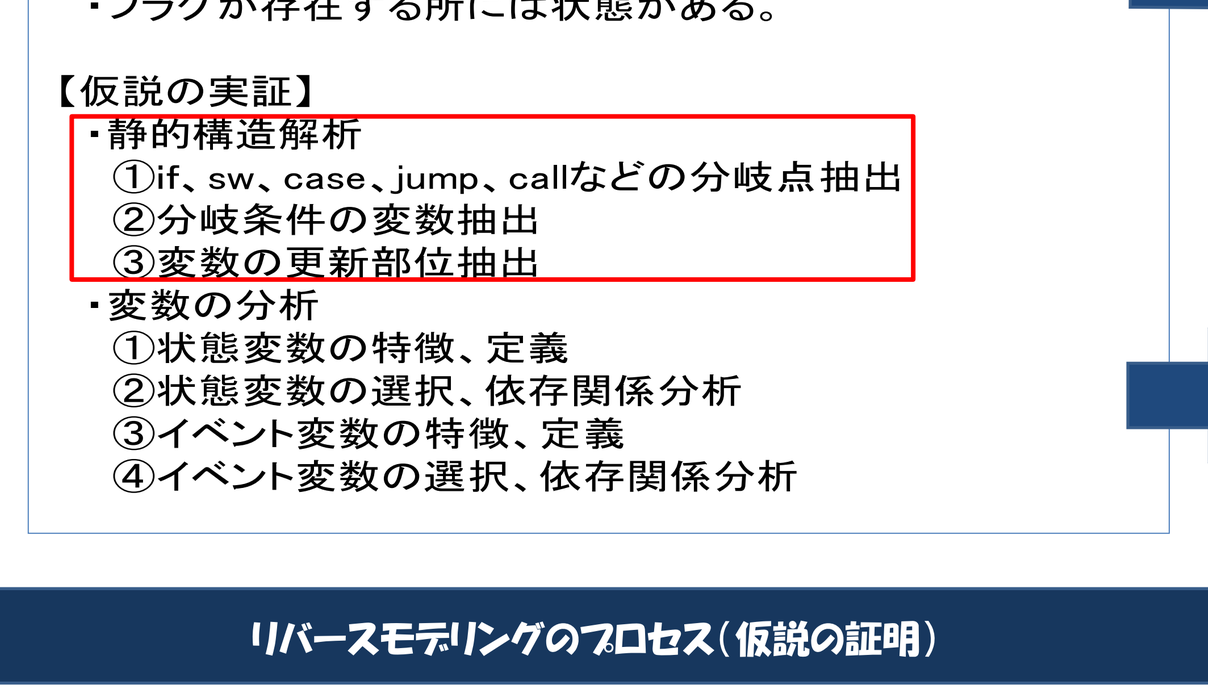
【仮説】
・フラグが存在する所には状態がある。

【仮説の実証】
・静的構造解析
①if、sw、case、jump、callなどの分岐点抽出
②分岐条件の変数抽出
③変数の更新部位抽出
・変数の分析
①状態変数の特徴、定義
②状態変数の選択、依存関係分析
③イベント変数の特徴、定義
④イベント変数の選択、依存関係分析

■ 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プロセス（仮説の証明）



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)

一般社団法人
組込みシステム技術協会
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)

Step3

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)

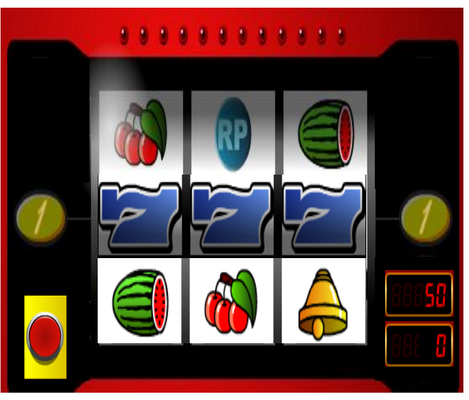
Step4

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

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。

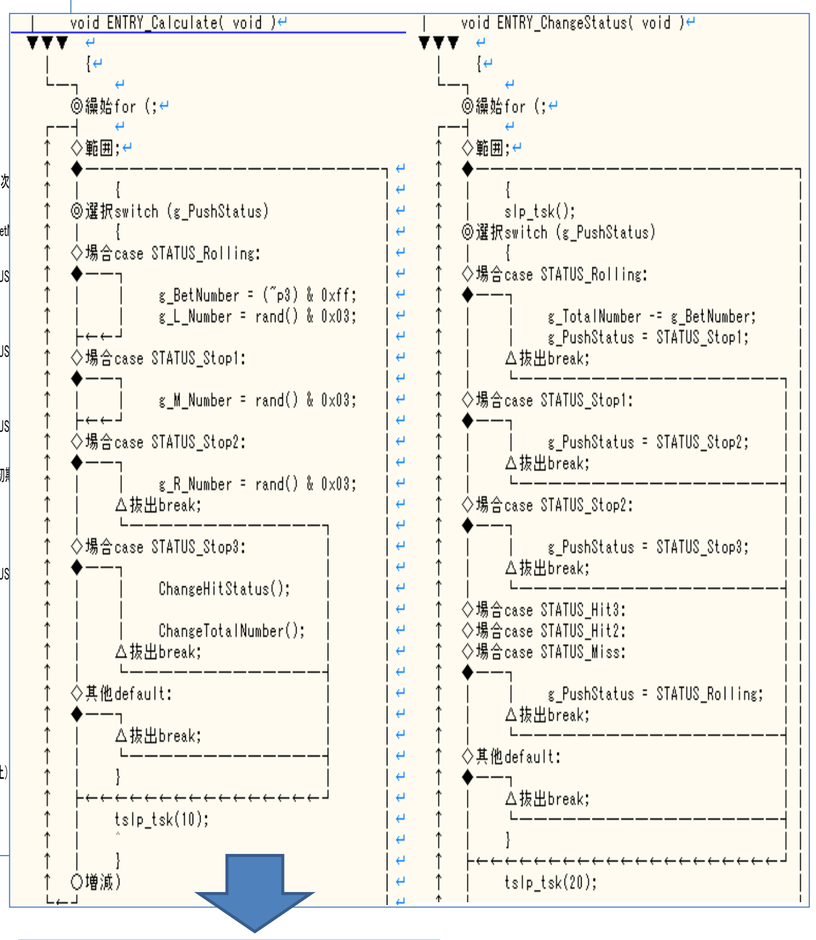
・4回目のINT押下で、すべてrolling状態とする

```
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
                        case STATUS_Rolling:
                                g_TotalNumber -= g_Bet
Bet */
                                g_PushStatus = STATUS
                        break;
                        case STATUS_Stop1:
                                g_PushStatus = STATUS
                        break;
                        case STATUS_Stop2:
                                g_PushStatus = STATUS
                        break;
                        /* 当たり判定済みの場合は ROLLING (初期
                        case STATUS_Hit3:
                        case STATUS_Hit2:
                        case STATUS_Miss:
                                g_PushStatus = STATUS
                        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1

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2

関数	条件	flag_b1_c			
		0	1	2	3
Entry4	LMR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;
	LMR 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;
	LMR 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3

関数	条件		int1cnt			
			0	1	2	3
entry3	◇total >= bet		total -= bet; int0ic = INT0_NG; int1cnt++;	X	X	X
	else	◇ranNum1 == ranNum3	int1cnt++;	int1cnt++;	int1cnt++; total += bet * 5;	int0ic = INT0_OK; int1cnt = 0; betFlag = ENABLE;
		◇ranNum1 == ranNum2 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

状態遷移表のリバースモデリングへの適用効果の考察

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。