



レガシーコードの蘇生術 ～リバースモデリングツール RExSTM for Cのご紹介～

2019年2月6日

状態遷移設計研究WG

山本 椋太

状態遷移設計研究会とは



http://www.jasa.or.jp/top/activity/state_transition.html

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

リンク集 | サイトマップ | プライバシーポリシー | お問い合わせ | English

協会案内 | 会員一覧 | 行事 | 協会活動 | 支部 | 会員専用

Home > 委員会活動 > 技術本部 > 技術高度化委員会 > 状態遷移設計研究会

委員会活動 技術本部 技術高度化委員会

状態遷移設計研究会

状態遷移設計研究会



【状態遷移設計研究会
主査】
青木 奈央
キャッツ(株)

平成28年度事業
既存ソースコードから、状態変数を抽出し状態遷移表をリバース生成する手法の研究を継続する。
・リバースモデリング手順のガイドの作成とツール化の検討。
・セミナー、講演会などの広報活動。他
なお、今年度は事例拡充のための、プロトタイプツールの作成を、産学連携（情報技術人材育成のための実践教育ネットワーク形成事業：e nPiT）により推進する。

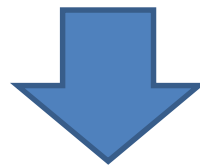
1. 定例会議
活動計画、進捗状況の確認を行う。
集中討議が必要な要件に対しては、合宿検討会を計画する。年1回を予定する。

2. セミナー、各種団体との交流

技術本部
人育成事業本部
事業推進本部
ET事業本部
OpenEL国際標準化委員会
プラグフェスト実行委



1. ツールRExSTM for Cの普及啓蒙、「状態遷移表のリバースモデリングへの適応」の研究
 - JASA会員内外への公開(フィードバックを募集、会員の1割を目指す。)
 - フィードバックのリスト化
 - ツールの改修
 - オープンソース化を目指す(ソースの整理、ルールの検討、アンケート内容検討など)



RExSTMの開発



■ 組み込みソフトウェア開発の傾向

- 派生開発による短納期・高品質の要望
- レガシーコードの肥大化・複雑化→メンテナンス性低下
→ 設計資料なし、担当者もすでにいない、
修正したら予想外の問題が出る…

ソースコードのブラックボックス化が進行中！



■ 効率化のための手法導入が進まない

- 派生開発・レガシーコードのせいで従来のものを踏襲せざるを得ない…

レガシーコードの存在が足かせになっている！





- リバースエンジニアリング
 - レガシーコードを解析することで仕様を明らかにする
- 仮説
 - 状態遷移設計は普遍的なモデル
 - レガシーコードにも状態遷移は必ずあるはず！



フラグのあるところに、状態がある！

- 研究テーマ
 - 「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遷移表をリバースモデリングする

状態遷移表でレガシーコードを蘇生！



状態遷移表



関数	条件	スロットの状態			
		全て回転中	1つ停止	2つ停止	3つ停止
Entry4	L、M、Rがすべて一致	状態遷移: 1つ停止	状態遷移: 2つ停止	状態遷移: 3つ停止 総額にBET金額の5倍を加算	状態遷移: 全て回転中
	L、M、Rの2つが一致	状態遷移: 1つ停止	状態遷移: 2つ停止	状態遷移: 3つ停止 総額にBET金額加算	状態遷移: 全て回転中
	else	状態遷移: 1つ停止	状態遷移: 2つ停止	状態遷移: 3つ停止	状態遷移: 全て回転中

状態変数

状態

イベント

遷移・処理

状態遷移表



関数	条件	flag_b1_c			
		0	1	2	3
Entry4	L、M、Rが すべて一致	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3 Total += Bet*5	flag_b1_c = 0
	L、M、Rの2つ が一致	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3 Total += Bet	flag_b1_c = 0
	else	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3	flag_b1_c = 0

状態変数

状態

イベント

遷移・処理

イベント

遷移・処理

サンプル検証1(意味不明→仕様明確化)



関数	条件	flag_b1_c			
		0	1	2	3
Entry4	L、M、Rが すべて一致	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3 Total += Bet*5	flag_b1_c = 0
	L、M、Rの2つ が一致	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3 Total += Bet	flag_b1_c = 0
	else	flag_b1_c = 1	flag_b1_c = 2	flag_b1_c = 3	flag_b1_c = 0

状態変数名が意味不明

状態を0~3の直値で指定

条件は 状態2 のときのみ有効

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

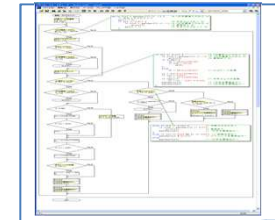
- コメント削除、#ifdefを設定
- 解析範囲の決定

```

char getStr()
{
    def(x=0,Y=0);
    def(x=1,Y=1);
    def(x=2,Y=2);
    def(x=3,Y=3);
    def(x=4,Y=4);
    def(x=5,Y=5);
    def(x=6,Y=6);
    def(x=7,Y=7);
    def(x=8,Y=8);
    def(x=9,Y=9);
    def(x=10,Y=10);
    def(x=11,Y=11);
    def(x=12,Y=12);
    def(x=13,Y=13);
    def(x=14,Y=14);
    def(x=15,Y=15);
    def(x=16,Y=16);
    def(x=17,Y=17);
    def(x=18,Y=18);
    def(x=19,Y=19);
    def(x=20,Y=20);
    def(x=21,Y=21);
    def(x=22,Y=22);
    def(x=23,Y=23);
    def(x=24,Y=24);
    def(x=25,Y=25);
    def(x=26,Y=26);
    def(x=27,Y=27);
    def(x=28,Y=28);
    def(x=29,Y=29);
    def(x=30,Y=30);
    def(x=31,Y=31);
    def(x=32,Y=32);
    def(x=33,Y=33);
    def(x=34,Y=34);
    def(x=35,Y=35);
    def(x=36,Y=36);
    def(x=37,Y=37);
    def(x=38,Y=38);
    def(x=39,Y=39);
    def(x=40,Y=40);
    def(x=41,Y=41);
    def(x=42,Y=42);
    def(x=43,Y=43);
    def(x=44,Y=44);
    def(x=45,Y=45);
    def(x=46,Y=46);
    def(x=47,Y=47);
    def(x=48,Y=48);
    def(x=49,Y=49);
    def(x=50,Y=50);
    def(x=51,Y=51);
    def(x=52,Y=52);
    def(x=53,Y=53);
    def(x=54,Y=54);
    def(x=55,Y=55);
    def(x=56,Y=56);
    def(x=57,Y=57);
    def(x=58,Y=58);
    def(x=59,Y=59);
    def(x=60,Y=60);
    def(x=61,Y=61);
    def(x=62,Y=62);
    def(x=63,Y=63);
    def(x=64,Y=64);
    def(x=65,Y=65);
    def(x=66,Y=66);
    def(x=67,Y=67);
    def(x=68,Y=68);
    def(x=69,Y=69);
    def(x=70,Y=70);
    def(x=71,Y=71);
    def(x=72,Y=72);
    def(x=73,Y=73);
    def(x=74,Y=74);
    def(x=75,Y=75);
    def(x=76,Y=76);
    def(x=77,Y=77);
    def(x=78,Y=78);
    def(x=79,Y=79);
    def(x=80,Y=80);
    def(x=81,Y=81);
    def(x=82,Y=82);
    def(x=83,Y=83);
    def(x=84,Y=84);
    def(x=85,Y=85);
    def(x=86,Y=86);
    def(x=87,Y=87);
    def(x=88,Y=88);
    def(x=89,Y=89);
    def(x=90,Y=90);
    def(x=91,Y=91);
    def(x=92,Y=92);
    def(x=93,Y=93);
    def(x=94,Y=94);
    def(x=95,Y=95);
    def(x=96,Y=96);
    def(x=97,Y=97);
    def(x=98,Y=98);
    def(x=99,Y=99);
    def(x=100,Y=100);
}
    
```

②構文ツリーの作成

- フローチャートの生成
- 構文ツリー図の作成



③条件処理表 (仮の状態遷移表) の作成

- 分岐条件を階層化
- 条件・処理の対応を記載した条件処理表の作成

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	State==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
	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)

④状態変数の抽出

- 状態変数の抽出
- 状態遷移表の編集

	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)

状態遷移表の作成 完了！



```
char c[128];
do{cE=cE/255;
do{cE=cE/255;
nc1=nc1/255; if ( nc1<0 ) nc1=;
hdcGetDC(hdc);
for( y=0;y<10;y+=YMAX; iCYMAX; y+=dx,i++)
for( x=0;x<10;x+=XMAX; x+=dx,i++,i++)
{
y=0; x=0;
//--設置調整--
for( i=0;i<10;i++)
{
x+=dx,y+=dy;
yB2=yB2+dy;
if ( xB2>=xB2+E ) break;
x+=dx; y+=dy;
}
c[i]=cE;
if ( c[i]>255 ) c[i]=255;
SetPixel(hdc,x,y,RGB(c[i],c[i],c[i]));
}
printf("C=NB,SIF C1=NB,SIF",c[i]);
TestOut(hdc,0,0,c[i]);
}
```

①レガシーコードの整形

- コメント削除、#ifdefを設定
- 解析範囲の決定

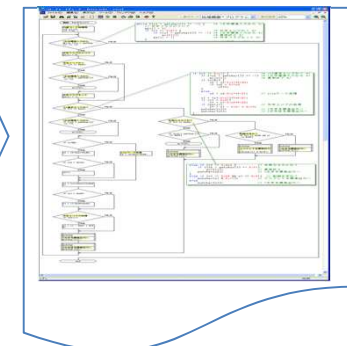
コメントはいらない！

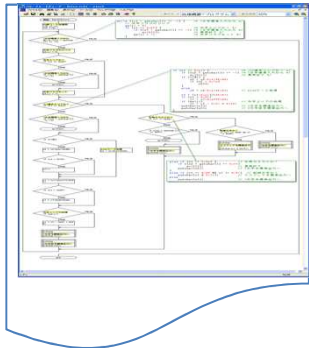
```
char c[128];
do{cE=cE/255;
do{cE=cE/255;
nc1=nc1/255; if ( nc1<0 ) nc1=;
hdcGetDC(hdc);
for( y=0;y<10;y+=YMAX; iCYMAX; y+=dx,i++)
for( x=0;x<10;x+=XMAX; x+=dx,i++,i++)
{
y=0; x=0;
//--設置調整--
for( i=0;i<10;i++)
{
x+=dx,y+=dy;
yB2=yB2+dy;
if ( xB2>=xB2+E ) break;
x+=dx; y+=dy;
}
c[i]=cE;
if ( c[i]>255 ) c[i]=255;
SetPixel(hdc,x,y,RGB(c[i],c[i],c[i]));
}
printf("C=NB,SIF C1=NB,SIF",c[i]);
TestOut(hdc,0,0,c[i]);
}
```

```
char c[128];
do{cE=cE/255;
do{cE=cE/255;
nc1=nc1/255; if ( nc1<0 ) nc1=;
hdcGetDC(hdc);
for( y=0;y<10;y+=YMAX; iCYMAX; y+=dx,i++)
for( x=0;x<10;x+=XMAX; x+=dx,i++,i++)
{
y=0; x=0;
//--設置調整--
for( i=0;i<10;i++)
{
x+=dx,y+=dy;
yB2=yB2+dy;
if ( xB2>=xB2+E ) break;
x+=dx; y+=dy;
}
c[i]=cE;
if ( c[i]>255 ) c[i]=255;
SetPixel(hdc,x,y,RGB(c[i],c[i],c[i]));
}
printf("C=NB,SIF C1=NB,SIF",c[i]);
TestOut(hdc,0,0,c[i]);
}
```

②構文ツリーの作成

- フローチャートの生成
- 構文ツリー図の作成





③条件処理表 (仮の状態遷移表) の作成

- 分岐条件を階層化
- 条件・処理の対応を記載した条件処理表の作成

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	State==S1 (Out=S)
	else (Out=0)

- 条件分岐をif-else構造でモデル化
- switch-caseもif-else構造
- 条件処理表の作成

リバースモデリングの作業手順

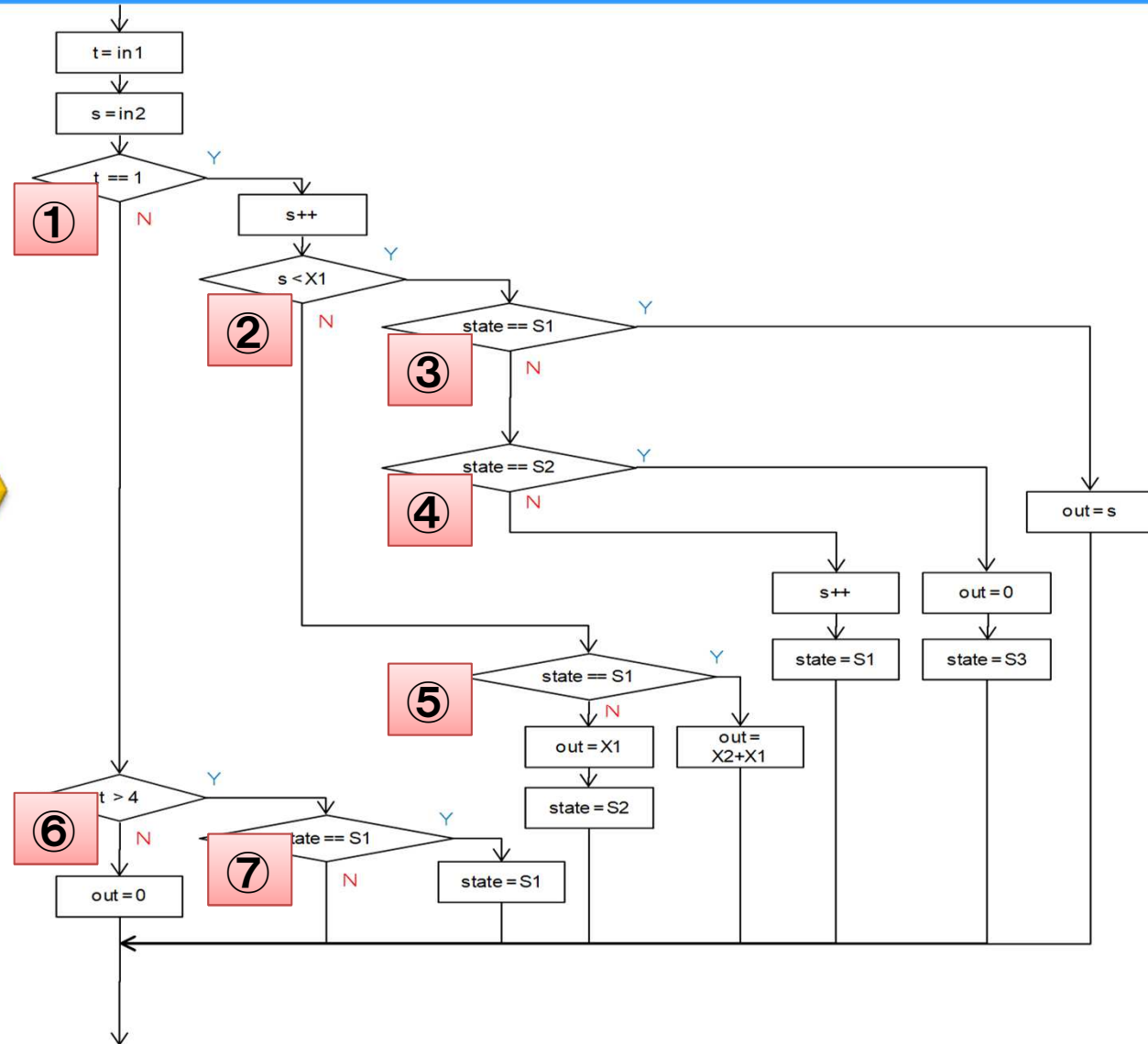


```

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;
}
}
    
```



リバースモデリングの作業手順



条件処理表の作成

条件		処理	
無条件実行		T=in1,S=in2	
T==1	①	S++	
	S<X1	②	flg==S1
		③	(Out=S)
		④	flg==S2
		else	S++flg,=S1
	else	⑤	flg==S1
		else	(Out=X2+X1)
T>4	⑥	flg==S1	
	⑦	else	(Out=X1), flg=S2
		(Out=S)	
		else	(Out=0)

ソースの
分岐条件から
条件を左、
処理を右、
に置いた仮の
状態遷移表を
作成

フローチャートの
①～⑦の構成と、
条件処理表の
①～⑦の構成は
同じになる



状態変数の抽出

状態遷移表の作成

条件	処理
無条件実行	T=in1.S=in2 S++
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)

④ 状態変数の抽出

- 状態変数の抽出
- 状態遷移表の編集

	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
	(Out=X2+X1)	(Out=X1).State=S2	(Out=X1).State=S2
T>4	(Out=S)	/	/
else	(Out=0)	(Out=0)	(Out=0)

■ ポイントは、状態変数を見つけること

■ 分岐条件の変数は状態変数か？

• 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



リバースモデリングの作業手順



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

flgはS1,S2,S3
の有限値
かつ
flgは
内部で更新
されている



flgは、
状態変数だ

elseを有限値
に置換

リバースモデリングの作業手順



状態遷移表の作成

条件		flg		
		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),flg=S3	/
		/	/	flg=S1
	else	(Out=X2+X1)	/	/
		/	(Out=X1),flg=S2	/
		/	/	(Out=X1),flg=S2
T>4	(Out=S)	/	/	
	/	/	/	
	/	/	/	
else		(Out=0)	(Out=0)	(Out=0)

flgを
状態変数
として、状態に
移行する

リバースモデリングの作業手順



条件		flg		
		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),flg=S3	flg=S1
	else	(Out=X2+X1)	(Out=X1),flg=S2	(Out=X1),flg=S2
T>4		(Out=S)	/	/
else		(Out=0)	(Out=0)	(Out=0)

うん、
状態遷移表に
なった



これで、やっとレビューができる！！



手作業で抽出することは非常に大変



- しかし、これらの手順を手作業で進めることは、非常にコストが大きい
 - 100行のソースコードでも2時間以上かかる
- それでもソースコードを理解するためには状態遷移表がほしい…

状態遷移表の抽出を自動化できないか？

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

- ・コメント削除、#ifdefを設定
- ・解析範囲の決定

②構文ツリーの作成

- ・フローチャートの生成
- ・構文ツリー図の作成

③条件処理表(仮の状態遷移表)の作成

- ・分岐条件を階層化
- ・条件・処理の対応を記載した条件処理表の作成

④状態変数の抽出

- ・状態変数の抽出
- ・状態遷移表の編集

状態変数の選択を
自動化することは
非常に難しい

状態遷移表の作成 完了!



- ソースコードから条件処理表は自動で抽出できそう

- 状態変数の選択は、ユーザ任せにしたい
 - どの変数を選択すると、意味のある状態遷移表になるか
 - でも状態変数を選ぶこともコストが大きい

- 状態変数候補を自動的に求め、ユーザが状態変数を選択する補助を行えるようにする
 - 状態変数の条件を満たしている変数を見つけてくる
 - ソースコードでどのように使用されているかを見たい



- 使いやすさを考え、インターフェースは Microsoft Office Excel とした
- 様々な機能拡張をするべく、コード解析などの処理は、Python で実装した
 - py2exe によって実行形式ファイルとし、それをExcelから呼び出している



■ 条件式(if/elseif/else、switch) と処理を区別

```
if(a == 1){                //タグ: IF1
    b = 1;                 //タグ: ST1
    c = 2;                 //タグ: ST2
}
else if(a == 2)           //タグ: IF2
    d = 0;                 //タグ: ST3
```

■ 条件式に依存する処理を抽出

```
if(a == 1) <= [b = 1;, c = 2;]
else if(a ==2) <= [d = 0;]
```

■ 依存関係を表現

```
IF1:ST1, ST2
IF2:ST3
```



- 条件式内に存在する変数を抽出
 - ・ 階層構造を利用し、その条件式で使われている変数が、更新されているかどうかを判定

```
if(a == 1){
    a = 0;
    b = 1;
    c = 2;
}
else if(a == 2) d = 0;
if(b == 3)c = 0;
```



```
if(a == 1){
    a = 0;
    b = 1;
    c = 2;
}
else if(a == 2) d = 0;
if(b == 3)c = 0;
```

a は値が更新される
→状態変数候補

b は値が更新されない
→状態変数候補でない

- 有限かどうかについては、判定していない
 - ・ 「有限」の定義が曖昧
 - ・ 現状のツールでは、Excelの表現可能な行/列数が限界



- 状態遷移表抽出器の実装
 - 手法とイメージが変わらないため、省略する

RExSTM for C 条件処理表・状態遷移表作成ツール



RExSTMver1_4.xlsm - Excel

ファイル ホーム 挿入 ページレイアウト 数式 データ 校閲 表示 開発 RExSTM

条件処理表作成 For C 状態変数選択フォーム シート削除共通

C12

	B	C	D	E	F	G	H	I
1								
2		条件処理表作成時の注意点						
3		(1) TargetProgramフォルダにある全てのcファイルが構文解析→条件処理表作成の対象となります。						
4		(2) 既にtempフォルダが存在する場合、temp内のファイルは全て削除されます						
5		(3) 同名のTSVファイルを入力した場合、古いコードが新しいコードに置き換わります。						
6		(4) 新しい条件処理表は、現在表示しているメイン操作画面シートのうしろに作成されます。						
7		(5) Windows8.10ユーザーの方へ:SmartScreenの機能により、ボタンを押しても処理が実行されないことがあります。						
8		ユーザマニュアルの「5.使用上の注意事項」に記載されている手順に従い、SmartScreenを無効にしてから実行してください。						
9		(6) リボンのRExSTMタブより、実行してください。						

メイン操作画面

準備完了 100%

ソースコードの整形・解析を行い、
状態遷移表生成のための情報整理を実施

RExSTM for C 状態遷移表の出力



RExSTM_demo.xlsxm - Microsoft Excel

	A	B	C	D	E	F	G	H	I	J
2										
3										
4							g.PushStatus			
5				0	1	2	3	4	5	6
6							g.PushStatus = 4;			
7			if((g_L_Number==g_M_Number)&&(g_M_Number==g_R_Number))	-	-	-	g.TotalNumber +=((g_BetNumber << 2)+g_BetNumber);			
8		void ENTRY_Calculate (void)	elseif((g_L_Number==g_M_Number) ((g_L_Number==g_R_Number) ((g_M_Number==g_R_Number)))	-	-	-	g.PushStatus = 5;			
9							g.TotalNumber += g_BetNumber;			
10			else	-	-	-	g.PushStatus = 6;			
11		void ENTRY_ChangeStatus(void)	無条件	g.PushStatus = 1;	g.PushStatus = 2;	g.PushStatus = 3;	-	g.PushStatus = 0;	g.PushStatus = 0;	g.PushStatus = 0;
12										
13										

実験: 状態遷移表のリバース 手動 VS 自動



名古屋大学の学生4名によるソースコードから状態遷移表を手動VS支援ツールで作成する実験を実施した。

実験対象ソースコード: 自動車の動作の一部を表すプログラムLOC110
(プラットフォームや開発環境向けに使用することが可能なコード)
電子レンジの動作の一部を表すプログラムLOC190
(RTOS(TOPPERS/ASP)のアプリケーションを意識したコード)

被験者	修正回数	経過時間
学生1	0回	19分
学生2	0回	12分
学生3	0回	10分
学生4	0回	20分

支援ツールを利用する場合

被験者	修正回数	経過時間
学生1	1回	91分
学生2	2回	108分
学生3	4回	85分
学生4	4回	122分

手動で行う場合

- 手動の場合は、間違いや見落とし等も発生しやすく修正回数が発生し、ツールを利用する場合と比べて合計秒数も5倍から6倍程度かかる
- ソースコードと状態遷移表の対応がとれていない
- 関数が不足している
- 表現するべきではない箇所についても状態遷移表を作成している



① ツール化活動

- ツール公開準備、不具合修正、マニュアル整備、機能の充実、ダウンロード時のアンケート検討、ライセンス記述の検討など

② 普及活動

- 展示会・セミナー・学会等での広報・宣伝活動、Bulletin JASAへの投稿、社内報への掲載など

③ ガイドライン

- ツールのプロモーション用の動画を作成

④ 限定リリース

- JASA会員向けの試用目的の限定リリース実施中



■ リバースモデリングツールの検討・作成

- 2014年度はツール要件固めを進めた
 - ー 自動でできる作業はツールで実行
 - ✓ ソースから条件・処理対応表(条件処理表)を生成
 - ✓ 状態変数候補を抽出、選択→状態遷移表に展開
 - ー 状態変数の特定など人力が必要な作業に集中
 - ✓ 現場で導入がしやすくなる、検証してもらえる
- 2015年度はenPiT を利用して実際にツール化
 - ー あるソースコードに対しては適用可能であることを確認した
- 2016年度はツールのブラッシュアップや、解析可能な対象を増やし、公開に向けての準備作業をした
 - ー マニュアルやドキュメント類を作成
 - ー 実際に使用したサンプルをツールにかけた

②普及活動 ③ガイドライン



- 2017年度は、ツールを公開し普及活動をする
 - ツールのプロモーション動画を作成しツールを利用したい企業へ配布
 - →配布することはできなかったが、展示会で公開し、多くの人に興味をもってもらうことができた
 - レガシーコードに問題がある企業等を訪問しヒアリングを行う
 - →訪問することはできなかったが、公開後にアンケートや訪問調査をすることに変更した
 - 論文や記事などを執筆する
 - →学会で発表、Bulletin JASAなどへ執筆掲載した
 - ツール公開後も不具合や使い勝手などを改修、サポートを実施
 - →サポートについては要検討、最終的にはオープンソースソフトウェア化する

④ 限定リリース



- 2018年8月より、JASA会員限定で公開を始めた。
 - 試用していただくことを目的としており、質問やパブリックコメントを頂いている。
- コメントを反映させられるよう、ツール修正の検討・修正作業を行っている。
- 修正版のツールをリリースした。

修正例)

- コンパイルオプションの指定に関する不具合
- 構文解析上の不具合



■ ソースコード→状態遷移表生成の多くを自動化

- 単純作業が減り、人が考える部分に集中できる
 - ー ソースコード整形、状態遷移表作成などは自動で実施
 - ー 手動と比べて作成時間を大きく減らすことができた
 - ー ただし、単純なパターンのみ

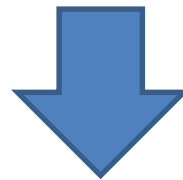
■ 状態変数の候補を自動抽出

- 状態変数の条件に合致する変数を自動抽出
- 候補として表示されたものから選ぶだけでよい
 - ー 手作業の場合、状態変数の特定が一番難しい

WGの議論中や開発中に気が付いたこと



- 実装者としてC言語の多さに改めて気が付いた。
- 要求仕様に時間がかかった
 - あいまいな部分を形にするのが難しかった
- 他の言語と比較しても、改めてバリエーションの豊富さに気が付いた
- 状態遷移表に落とし込むときに非常に苦労した
- 状態遷移設計に対するバリエーションの多さが難しかった
- 理論的に理解しても、それをロジックに落とし込むときに想定していないコードがあった
- 状態遷移表の構造との差分がコードに見受けられた
- 状態モデルで作成されたソースコードでも後日パッチ等を当てた場合に、当初の状態モデルのコードがくずれてしまい、そのようなソースコードのリバーズには苦労した。
- 状態変数を見つけることじたいが非常に困難



状態遷移設計をされているコードをみつけることだけでも意味がある

公開についての予定



■ 公開の日程

- 現在、JASA会員向けに公開中です！
- 1ヶ月の試用期間で試用いただき、コメントを頂いております
- 12月末まで募集しています

■ パブリックコメント(2018年12月末で終了しました)

- ツールの使い方や不具合等の質問を受け付けます
- 公開は、ダウンロードする方の社名や連絡先を記入していただき、ダウンロードページへ移動できるような仕組みをとります

■ ツールの修正

- パブリックコメントをベースにツールを改修する
- パブリックコメント目標は20件とし、改修可能かどうかを検討後、オープンソースに向けてソースコードを整理し、改修スケジュールを作成する

■ ET-WEST 2019 を目標にオープンソースソフトウェアとして公開する予定

ツール開発としての結論



- 2014年の議論開始当初は、ツールを販売する？コンサルを行う？ことなどが議論されていた
- 現状のツールは、そのまま販売できるまでに至っていない
 - C言語の多様性を考えるとそのままでの販売は困難→オープンソース化して公開
- ツールを利用したコンサルの可能性
 - WGのメンバーは、所属企業があるためコンサルをずっと実施するためには、所属企業を巻き込む必要があり、実施するためには時間が必要になるのですぐには困難
 - ハンズオンセミナー等の実施を検討する必要がある
- 当初は、ツールの多言語化対応を目指して開発を進めていた
 - 組込みシステムで多く利用されているC言語から状態遷移表を抽出するツールに着手した
 - 今後は、オープンソース化をして幅広い組込み分野で使用できるようにしていく
- 今回のツール開発は、プログラムの構造化(状態変数候補の抽出)を可視化できたところでは非常に意味のあるツールができたと考える





ご清聴ありがとうございました

ツールは、JASA会員様に無償にて配布させていただきます。詳しくは下記にご連絡をお願い致します。

【問い合わせ先】状態遷移設計WG 担当者

E-mail: jasainfo@jasa.or.jp

TEL: 03-5643-0211

HP: <http://www.jasa.or.jp/TOP>



【レガシーコードの蘇生術】

2019/2/6 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大伝馬町6-7
TEL: 03 (5643) 0211 FAX: 03 (5643) 0212
URL: <http://www.jasa.or.jp/>

本書の著作権は一般社団法人組込みシステム技術協会（以下、JASTA）が有します。
JASTAの許可無く、本書の複製、再配布、譲渡、展示はできません。
また本書の改変、翻案、翻訳の権利はJASTAが占有します。
その他、JASTAが定めた著作権規程に準じます。