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～

2018年7月5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

状態遷移設計研究WGの方針

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)
2. ビッグデータ関連のCEPについて研究

RExSTMの開発

現在の組込みソフト開発の状況

■ 組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！

仮説と研究テーマ

■ 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

サンプル検証1（意味不明→仕様明確化）

状態変数名が意味不明

状態を0～3の直値で指定

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

条件は 状態2 のときのみ有効

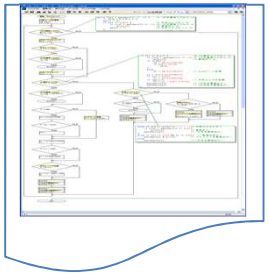
リバースモデリングの作業手順

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

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	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
		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

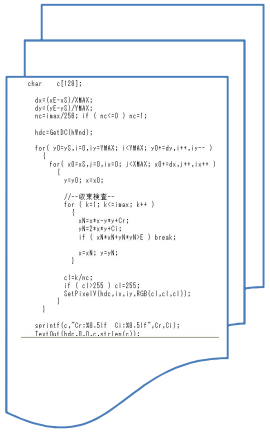
- 状態変数の抽出
- 状態遷移表の編集

		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！

リバースモデリングの作業手順

ソースコード前処理



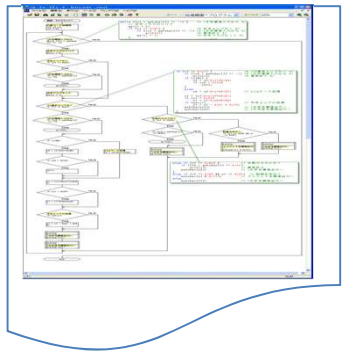
①レガシーコードの整形

・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！

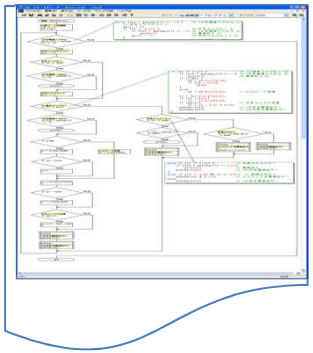
②構文ツリーの作成

・フローチャートの生成
・構文ツリー図の作成



リバースモデリングの作業手順

条件処理表の作成



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

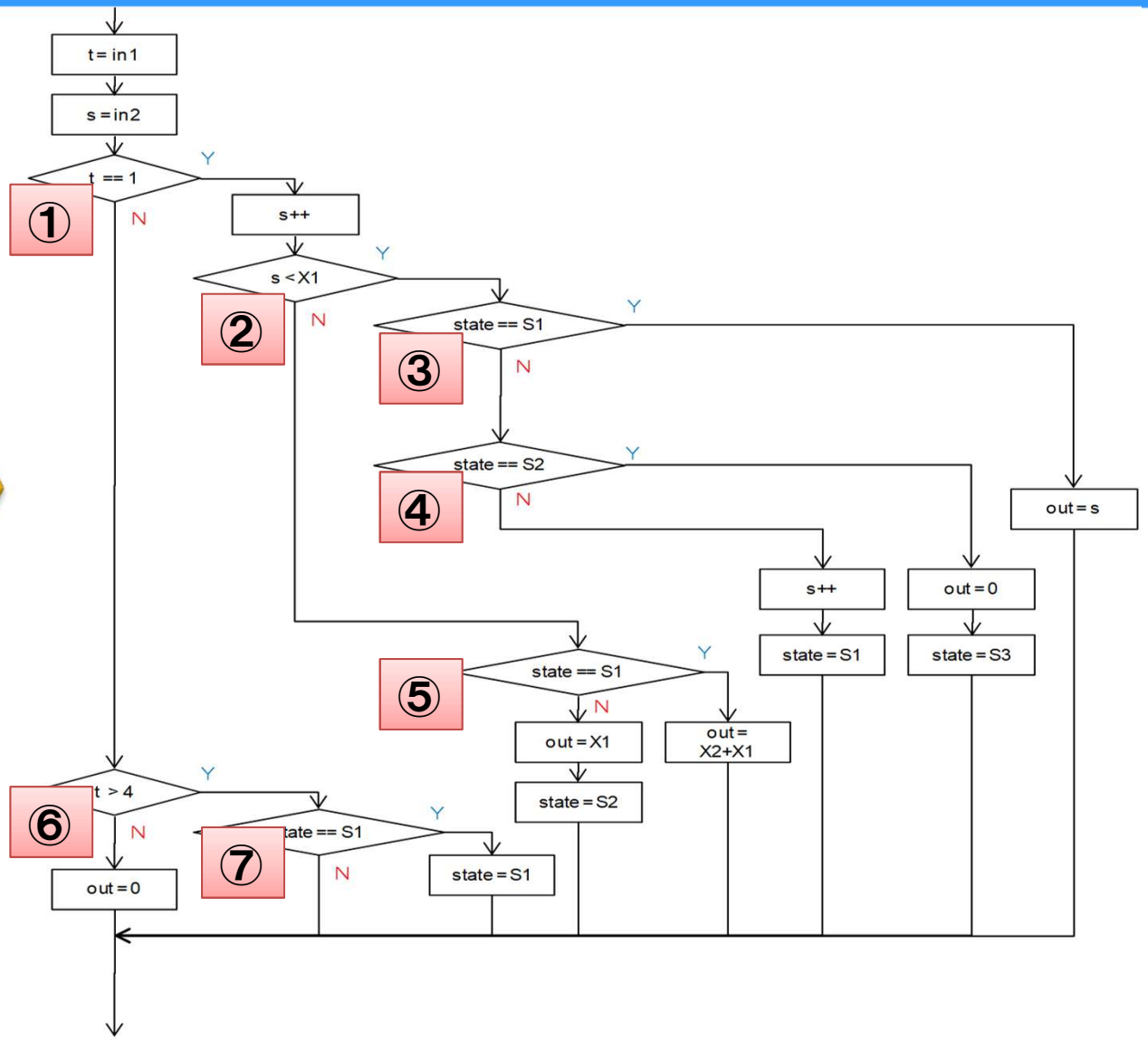
・条件分岐をif－else構造でモデル化
・switch-caseもif－else構造
・条件処理表の作成

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)
		flg==S2 ④	(Out=0), flg=S3
		else	S++flg,=S1
	else	flg==S1 ⑤	(Out=X2+X1)
		else	(Out=X1), flg=S2
T>4 ⑥	⑦	flg==S1	(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

リバースモデリングの作業手順

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
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)

■ 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

- ただ，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！

ツール化の方針

■ 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

ツール化の方針

- 使いやすさを考え，インタフェースは 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.

ツールの実装：ソースコード解析

■ 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
```

ツールの実装：状態変数候補抽出器

■ 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

ツールの実装

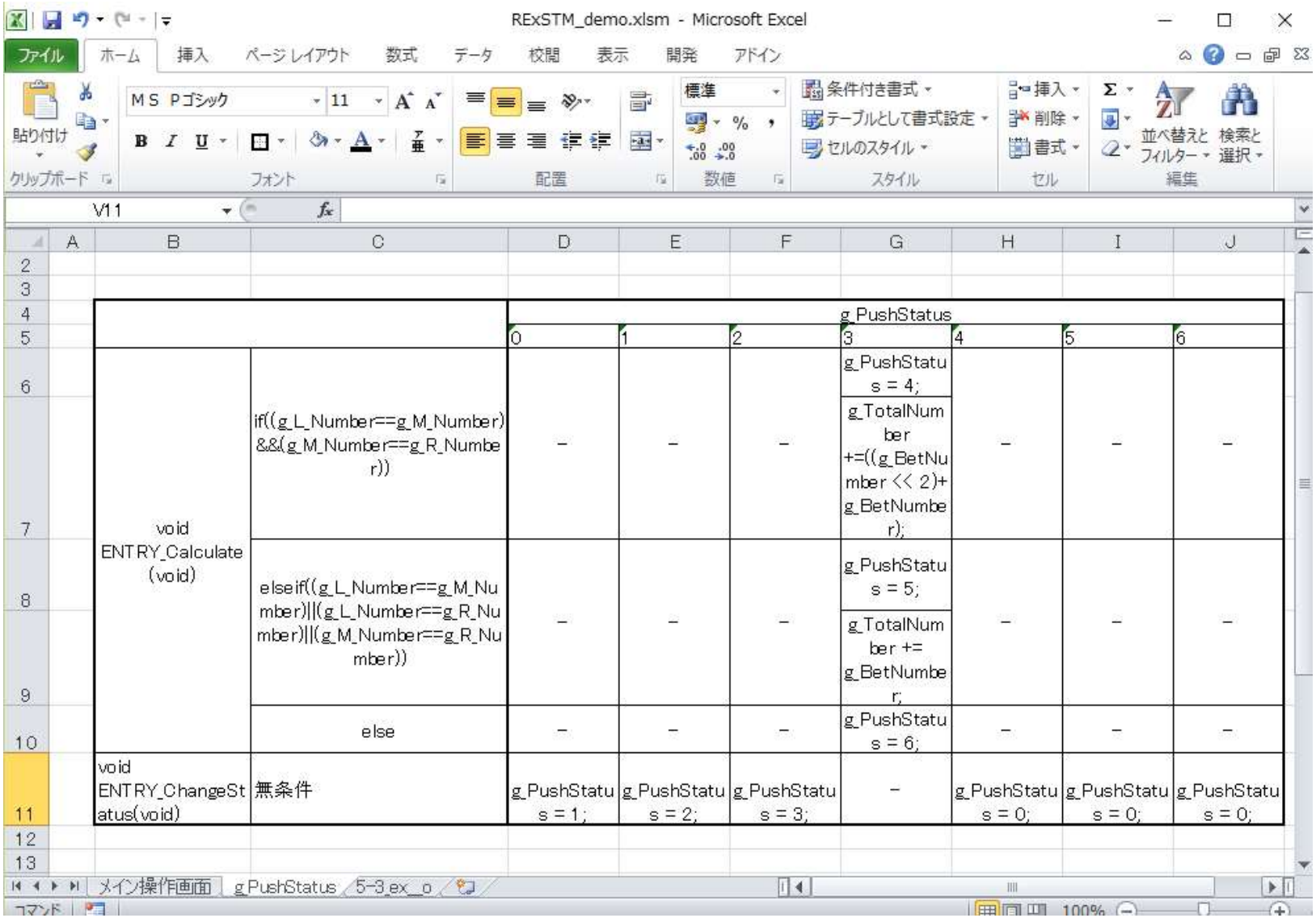
■ 状態遷移表抽出器の実装

- 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

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



			g_PushStatus						
			0	1	2	3	4	5	6
void ENTRY_Calculate (void)	if((g_L_Number==g_M_Number) &&(g_M_Number==g_R_Numbe r))		–	–	–	g_PushStatu s = 4; g_TotalNum ber +=((g_BetNu mber << 2)+ g_BetNumbe r);	–	–	–
	elseif((g_L_Number==g_M_Nu mber)||(g_L_Number==g_R_Nu mber)||(g_M_Number==g_R_Nu mber))		–	–	–	g_PushStatu s = 5; g_TotalNum ber += g_BetNumbe r;	–	–	–
	else		–	–	–	g_PushStatu s = 6;	–	–	–
void ENTRY_ChangeSt atus(void)	無条件		g_PushStatu s = 1;	g_PushStatu s = 2;	g_PushStatu s = 3;	–	g_PushStatu s = 0;	g_PushStatu s = 0;	g_PushStatu s = 0;

ツールデモ

■　画面を御覧ください

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	合計秒数
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	合計秒数
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

2017年度　活動報告

① ツール化活動

- ツール公開に向けての準備、主に大きな不具合修正、マニュアルの整備、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

- 展示会での広報活動、セミナーや学会等での宣伝活動、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

→動画再生

①ツール化活動

■ 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
  - 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**
  - 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
  - 現状，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
  - マニュアルやドキュメント類を作成
  - 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などへ執筆掲載することができた
  - 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化する

RExSTM for C　今回の成果

■　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スには苦労した。
- 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

- 昨年9月に公開を延期したものの致命的な不具合が見つかり、修正やテスト等に時間を費やしたため、公開が今年度になってしまった。
- 理由：現状のものを公開するよりもより品質が高いものを公開するほうがよいと判断（現状の機能に関する不具合を修正する必要があるため）

■ パブリックコメント

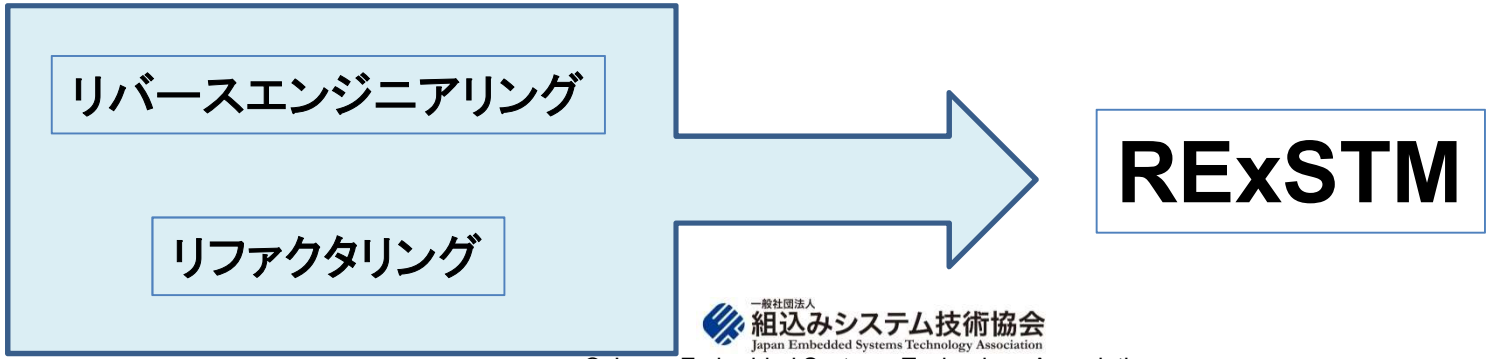
- ツールを公開後、期間を決めてJASAのメンバー専用ページで公開する
- 公開期間は3か月程度とし、ツールの使い方や不具合等の質問を受け付けます
- 公開は、ダウンロードする方の社名や連絡先を記入していただいて、ダウンロードページへ移動できるような仕組みをとります。
- 必要であれば、会社訪問を行いサポートを実施（会社訪問費用は計上済み）

■ 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

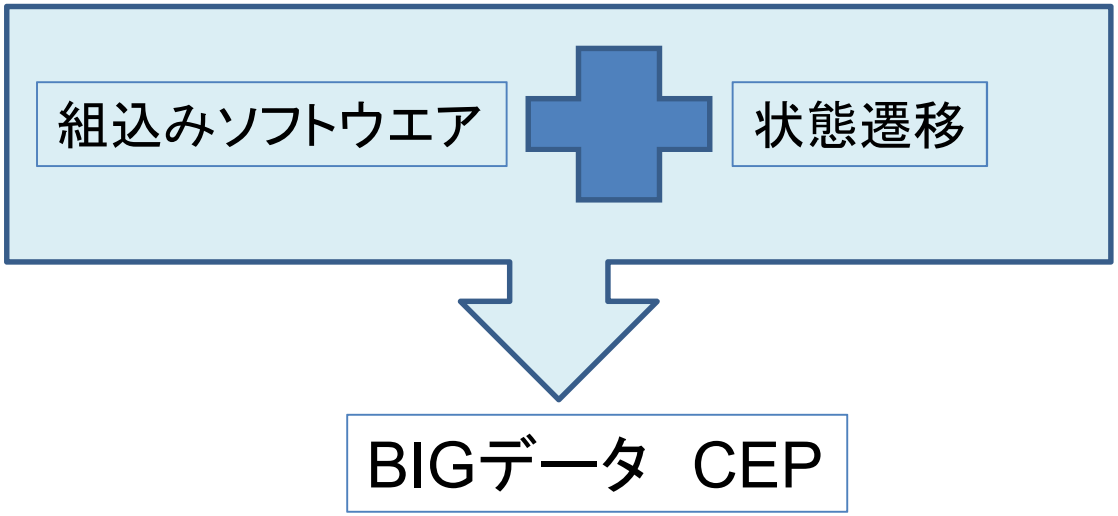
ツール開発としての結論

- 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



WGとしての今後

- 今後は、BIGデータ関連で、まだ日本に浸透していないCEP（Complex Event Processing）について勉強会を実施する



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頂きます。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】

2018/7/5 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/