オープンソースの組み合わせテストツールの開発

ー組み合わせにおける制約の表形式でのモデル化ー

岩通ソフトシステム株式会社
鶴巻敏郎　柳田幸秀

JaSST 2009 東京

組み合わせテストツールの現状

- 組み合わせ方式には直交表方式とPairwise(All-Pairs)方式がある
- どちらの方式が優れているか結論は出ていない
- ツールの発表は多いが公開されることはない
- テスト業務で使うには「制約」のサポートが必須条件

- 直交表方式で制約をサポートした実用的なツールは有償・無償含めて皆無
- 直交表方式で制約をサポートした実用的な組み合わせ生成エンジンを開発するには多額な費用が見込まれる
- Pairwise方式でも制約をサポートしたエンジンの開発にはそれなりの費用がかかる

Pairwise方式のフリーソフト

- Pairwise方式の有償・無償のソフトを集めたサイト
  Pairwise Testing　http://www.pairwise.org/

- フリーソフトで制約をサポートした優れた生成エンジンが見つかった　「PICT」と「Jenny」

- 但し生成エンジンそのままでは日本語が文字化けするなど**実用性に欠ける**

- 生成エンジンを実用的に使えるようにするソフトが必要
  エンジンを自主開発するより**大幅に少ない費用**で済む

開発した組み合わせテストツール

PictMaster

- 組み合わせ生成エンジンにPICTを使用
- パラメータ（因子）は30個まで　パラメータあたりの値（水準）は30個まで　制約指定は30個まで定義可能

AllPairII

- 組み合わせ生成エンジンにPICTのほかにJennyを追加
- パラメータ（因子）は80個まで　パラメータあたりの値（水準）は50個まで　制約指定は80個まで定義可能
- PictMasterの機能はそのままにJennyを追加したことによりPICTでは処理できない多数の制約を持つ対象を高速に処理可能

★ https://sourceforge.jp/projects/pictmaster からダウンロードできる

制約をどのように記述するか

- 「制約」とは組み合わせできない組み合わせのこと

  OSとブラウザとメールソフトの組み合わせでは

パラメータ	値の並び
OS	Windows, Mac OS, Linux
ブラウザ	IE, FireFox, Safari
メールソフト	Outlook, Thunderbird, Mail

OSがMac OSとLinuxではブラウザにIEは組み合わせ不可
OSがWindowsとLinuxではメールソフトにMailは組み合わせ不可

- この制約の記述形式は

PICT

```
if [OS] = “Mac OS” or [OS] =”Linux” then [ブラウザ] <> “IE”;
if [OS] = “Windows” or [OS] =”Linux” then [メールソフト] <> “Mail”;
```

Jenny

```
-w 1bc2a -w 1a3c
```

- 既存ツール　制約を１行づつ指定する一次元の行形式
  → 制約が多くなると制約間の関係把握が困難になる

制約を表現する用語の定義

- 制約を if 関係式 then 関係式 の形式で記述した式を制約式という
- 関係式はパラメータと値を比較するために = <> などを演算子とする
- 関係式は、（パラメータ 演算子 値）で構成される
- 演算子が = の場合を順制約といい、<> の場合を逆制約という
- ifとthenの間の関係式を制約条件 then以降の関係式を制約対象という

if 制約条件 then 制約対象

制約条件　あるパラメータが持つすべての値のうち制約条件の演算子との関係で真となる特定の値を指定する

制約対象　制約条件で指定された値と、制約対象の演算子との関係で組み合わせ可能な他のパラメータの値を指定する

制約表の使い方

- 開発したツールでは制約を指定する方法として「制約表」を用いる

制約表			
パラメータ	制約１	制約２	制約３

- パラメータの列には、１行ごとにパラメータの名称を記入する

- 制約１、制約２、・・・ の列には制約条件となる値と制約対象となる値を、対応するパラメータの行のセルに記入する

- 値だけを記入すると、演算子に = を指定したことを意味し、値の前に # を記述すると、演算子に <> を指定したことを意味する

- 制約欄に記入した値は同じ行のパラメータ欄に記入されたパラメータと演算子を介して特定の値を指定することになる

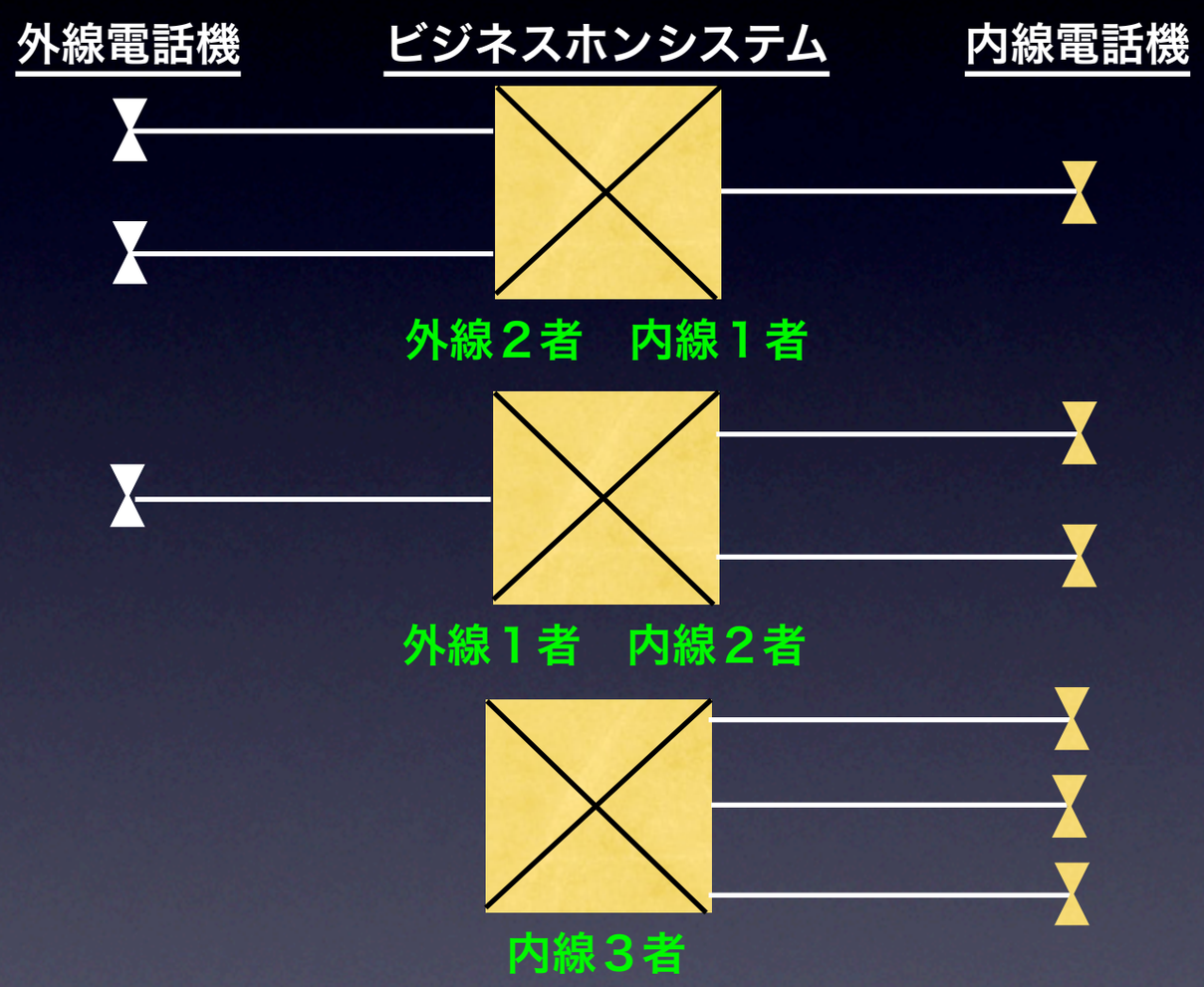
- 制約条件のセルは任意の色で塗りつぶす　制約対象のセルは塗りつぶしを行なわない

制約表を用いるメリット

- 既存の1次元の行形式と比較した制約表のメリット
  - 制約が表モデル形式でかつ色付きで視覚的に表現されるため、制約の意味が直感的に理解できる
  - 制約が2次元の行と列による表形式で俯瞰できるため、制約間の関係が理解しやすい
  - 状態遷移テスト、制御パステストのテストケースを簡単に作成できる
- PICTの制約式と比較した制約表のメリット
  - 原理的に構文エラーが生じない
  - 記述が簡単で記述量が大幅に少なくて済む
  - どの箇所でエラーとなったかが制約表のセルを指定することで明確に指摘できる

総合テストへの適用例

ビジネスホンシステムでの３者会議通話の組み合わせテストの例を示す
３者会議通話には、**通話種別**に以下の３つがある



このテストでは、通話種別の他に外線の回線種別と内線の端末種別が組み合わせテストのパラメータとなる　そして図に示すように通話種別によって接続の構成が異なる
従って通話種別が制約条件となり、回線種別と端末種別が制約対象となる

３者会議通話のモデルと制約表

パラメータ		値の並び	
回線		IP外線, ISDN, アナログ, 内線	
通話種別		外線２内線１, 外線１内線２, 内線３	
端末種別１		KT, DCL, SLT	
端末種別２		KT, DCL, SLT, -	
端末種別３		KT, DCL, SLT, -	

制約表

パラメータ	制約１	制約２	制約３
回線	#内線	#内線	内線
通話種別	外線２内線１	外線１内線２	内線３
端末種別１			
端末種別２	-	#-	#-
端末種別３	-	-	#-

端末2と端末3の値「-」はその値が意味をなさないダミーの値であることを示す。例えば通話種別が外線2内線1の場合、端末2と端末3は存在しないため、ダミーの値「-」を割り当てる。

```
if ([通話種別] = "外線２内線１" )
   then ([回線] <> "内線" ) and ([端末種別２] = "-" ) and ([端末種別３] = "-" ) ;
if ([通話種別] = "外線１内線２" )
   then ([回線] <> "内線" ) and ([端末種別２] <> "-" ) and ([端末種別３] = "-" ) ;
if ([通話種別] = "内線３" )
   then ([回線] = "内線" ) and ([端末種別２] <> "-" ) and ([端末種別３] <> "-" ) ;
```

ツールがモデルと制約表から生成した制約式

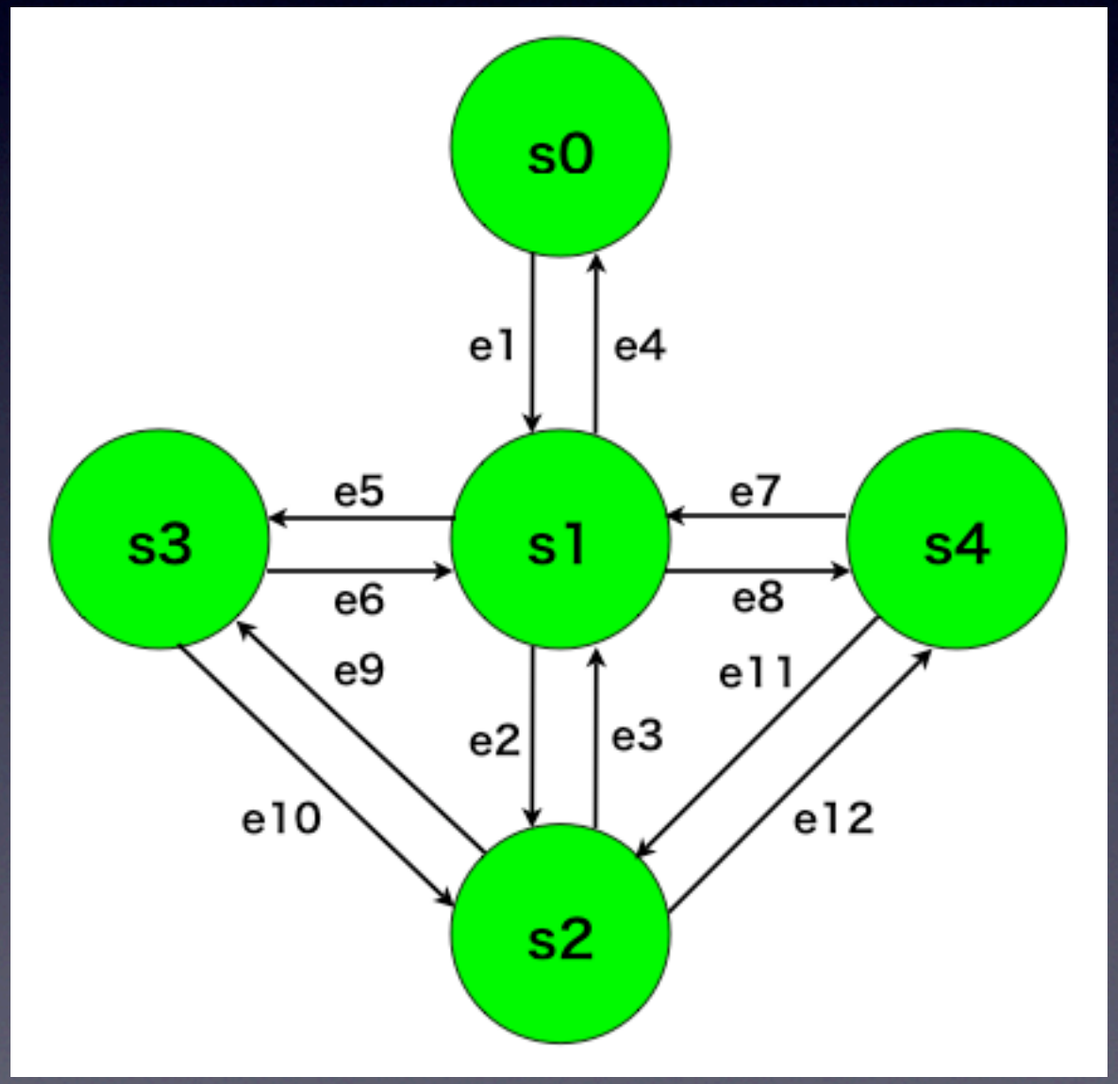
３者会議通話の組み合わせテストケース

No.	回線	通話種別	端末種別１	端末種別２	端末種別３
1	IP外線	外線１内線２	DCL	SLT	-
2	IP外線	外線１内線２	KT	KT	-
3	IP外線	外線１内線２	SLT	DCL	-
4	IP外線	外線２内線１	DCL	-	-
5	ISDN	外線１内線２	DCL	KT	-
6	ISDN	外線１内線２	DCL	SLT	-
7	ISDN	外線１内線２	KT	DCL	-
8	ISDN	外線２内線１	SLT	-	-
9	アナログ	外線１内線２	DCL	SLT	-
10	アナログ	外線１内線２	DCL	KT	-
11	アナログ	外線１内線２	SLT	DCL	-
12	アナログ	外線２内線１	KT	-	-
13	内線	内線３	DCL	DCL	KT
14	内線	内線３	DCL	KT	DCL
15	内線	内線３	DCL	KT	SLT
16	内線	内線３	KT	DCL	DCL
17	内線	内線３	KT	SLT	KT
18	内線	内線３	KT	DCL	SLT
19	内線	内線３	SLT	SLT	SLT
20	内線	内線３	SLT	KT	KT
21	内線	内線３	SLT	SLT	DCL

通話種別に応じて組み合わせ可能な端末の数が異なる

状態遷移テストへの適用例

図で示す状態s0～s4があり、イベントe1～e12で状態を遷移する
遷移可能なすべての状態の組み合わせをPairwise法で網羅することができる
状態遷移はs0から始まりs0で終わる
状態s1を最大3回まで通る遷移ルートをすべて生成する



状態遷移テストのモデルと制約表

パラメータは状態の往復を表現するため同じ状態にダッシュ（'）を付加している

値の名称はイベントと次の遷移先の状態を表している

制約表は制約対象が次の制約の制約条件になるという簡単なルールで記入する

パラメータ		値の並び
s0		e1-s1, -
s1		e4-s0, e2-s2, e5-s3, e8-s4, -
s2		e3-s1, e9-s3, e12-s4, -
s3		e6-s1, e10-s2, -
s4		e7-s1, e11-s2, -
s1'		e4-s0, e2-s2, e5-s3, e8-s4, -
s2'		e3-s1, e9-s3, e12-s4, -
s3'		e6-s1, -
s4'		e7-s1, -
s1''		e4-s0, -
s0'		end, -

制約表

パラメータ	制約１	制約２	制約３	制約４	制約５	制約６	制約７
s0	-	e1-s1					
s1	-	e4-s0, e2-s2, e5-s3, e8-s4	e4-s0	e2-s2	e5-s3	e8-s4	
s2	-		-	e3-s1, e9-s3, e12-s4	-	-	e3-s1
s3	-		-		e6-s1, e10-s2	-	-
s4	-		-			e7-s1, e11-s2	-
s1'	-		-				e4-s0, e5-s3, e8-s4, e2-s2
s2'	-		-				
s3'	-		-				
s4'	-		-				
s1''	-		-				
s0'	-		end				

状態遷移テストの制約表（その２）

制約８	制約９	制約１０	制約１１	制約１２	制約１３	制約１４	制約１５
e9-s3	e12-s4						
e6-s1, e10-s2	-	e6-s1	e10-s2				
	e7-s1, e11-s2	-	-	e7-s1	e11-s2		
		e4-s0, e2-s2, e8-s4, e5-s3	-	e4-s0, e2-s2, e5-s3, e8-s4	-	e4-s0	e5-s3
			e3-s1, e9-s3, e12-s4		e3-s1, e9-s3, e12-s4	-	-
						-	e6-s1
						-	
						-	
						end	

制約１６	制約１７	制約１８	制約１９	制約２０	制約２１	制約２２	制約２３
e8-s4	e2-s2						
-	e3-s1	e3-s1	e9-s3	e12-s4			
-		-	e6-s1	-	e6-s1		
e7-s1		-		e7-s1	-	e7-s1	
		e4-s0			e4-s0	e4-s0	
		end			end	end	

状態遷移テストケース

No.	s0	s1	s2	s3	s4	s1'	s2'	s3'	s4'	s1''	s0'
1	-	-	-	-	-	-	-	-	-	-	-
2	e1-s1	e2-s2	e12-s4	-	e7-s1	e5-s3	-	e6-s1	-	e4-s0	end
3	e1-s1	e2-s2	e12-s4	-	e11-s2	-	e12-s4	-	e7-s1	e4-s0	end
4	e1-s1	e2-s2	e12-s4	-	e7-s1	e8-s4	-	-	e7-s1	e4-s0	end
5	e1-s1	e2-s2	e12-s4	-	e7-s1	e2-s2	e3-s1	-	-	e4-s0	end
6	e1-s1	e2-s2	e12-s4	-	e7-s1	e4-s0	-	-	-	-	end
7	e1-s1	e2-s2	e12-s4	-	e11-s2	-	e9-s3	e6-s1	-	e4-s0	end
8	e1-s1	e2-s2	e12-s4	-	e11-s2	-	e3-s1	-	-	e4-s0	end
9	e1-s1	e2-s2	e3-s1	-	-	e4-s0	-	-	-	-	end
10	e1-s1	e2-s2	e3-s1	-	-	e5-s3	-	e6-s1	-	e4-s0	end
11	e1-s1	e2-s2	e3-s1	-	-	e8-s4	-	-	e7-s1	e4-s0	end
12	e1-s1	e2-s2	e3-s1	-	-	e2-s2	e3-s1	-	-	e4-s0	end
13	e1-s1	e2-s2	e9-s3	e6-s1	-	e4-s0	-	-	-	-	end
14	e1-s1	e2-s2	e9-s3	e6-s1	-	e8-s4	-	-	e7-s1	e4-s0	end
15	e1-s1	e2-s2	e9-s3	e10-s2	-	-	e12-s4	-	e7-s1	e4-s0	end
16	e1-s1	e2-s2	e9-s3	e10-s2	-	-	e3-s1	-	-	e4-s0	end
17	e1-s1	e2-s2	e9-s3	e6-s1	-	e2-s2	e3-s1	-	-	e4-s0	end
18	e1-s1	e2-s2	e9-s3	e10-s2	-	-	e9-s3	e6-s1	-	e4-s0	end
19	e1-s1	e2-s2	e9-s3	e6-s1	-	e5-s3	-	e6-s1	-	e4-s0	end
20	e1-s1	e4-s0	-	-	-	-	-	-	-	-	end
21	e1-s1	e5-s3	-	e6-s1	-	e2-s2	e3-s1	-	-	e4-s0	end
22	e1-s1	e5-s3	-	e6-s1	-	e5-s3	-	e6-s1	-	e4-s0	end
23	e1-s1	e5-s3	-	e6-s1	-	e4-s0	-	-	-	-	end
24	e1-s1	e5-s3	-	e10-s2	-	-	e9-s3	e6-s1	-	e4-s0	end
25	e1-s1	e5-s3	-	e10-s2	-	-	e12-s4	-	e7-s1	e4-s0	end
26	e1-s1	e5-s3	-	e6-s1	-	e8-s4	-	-	e7-s1	e4-s0	end
27	e1-s1	e8-s4	-	-	e7-s1	e8-s4	-	-	e7-s1	e4-s0	end
28	e1-s1	e8-s4	-	-	e7-s1	e4-s0	-	-	-	-	end
29	e1-s1	e8-s4	-	-	e7-s1	e2-s2	e3-s1	-	-	e4-s0	end
30	e1-s1	e8-s4	-	-	e7-s1	e5-s3	-	e6-s1	-	e4-s0	end
31	e1-s1	e8-s4	-	-	e11-s2	-	e9-s3	e6-s1	-	e4-s0	end
32	e1-s1	e8-s4	-	-	e11-s2	-	e12-s4	-	e7-s1	e4-s0	end

生成されたテストケースは、状態s1を最大3回通るすべての状態遷移ルートを網羅している

No.1のケースは１つのパラメータに値は２つ以上必要という生成エンジンの制限に合わせたもの　実際のテストケース数は31個

制約表への記入は簡単なルールに沿って行なう機械的な作業で済む

サブシステムで状態の往復がある状態遷移テストに有効

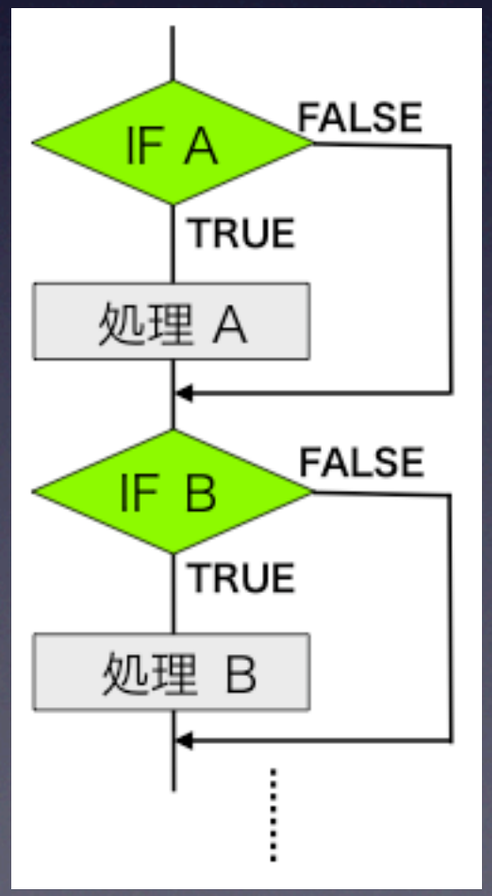
制御パステストへの適用

制御パステストの網羅基準

- 命令網羅　プログラムの命令文を網羅する
- 条件網羅　プログラムの条件文を網羅する
- 経路網羅　プログラムの経路を網羅する

制御パステストへのPairwise法の適用

すべての条件文の真偽両方のパスを網羅する条件網羅のテストケースが生成される



No.	IF A	IF B	IF C
1	FALSE	FALSE	TRUE
2	FALSE	TRUE	FALSE
3	TRUE	FALSE	FALSE
4	TRUE	TRUE	TRUE

IF文が３つのときテストケースは4個となる

基本的な条件網羅では左のIF文が10個あってもテストケースは2個である
これは制御パスの特殊なテストケース

Pairwise法を適用した制御パステストでは各IF文をパラメータとし、真偽2つの分岐先を値とした2パラメータ間の組み合わせとして制御パスが網羅されるため、IF文が10個のときは8個のテストケースとなる

制御パステストへの適用上の問題と解決策

ループを含むプログラムへ制御パステストを適用する上での問題点

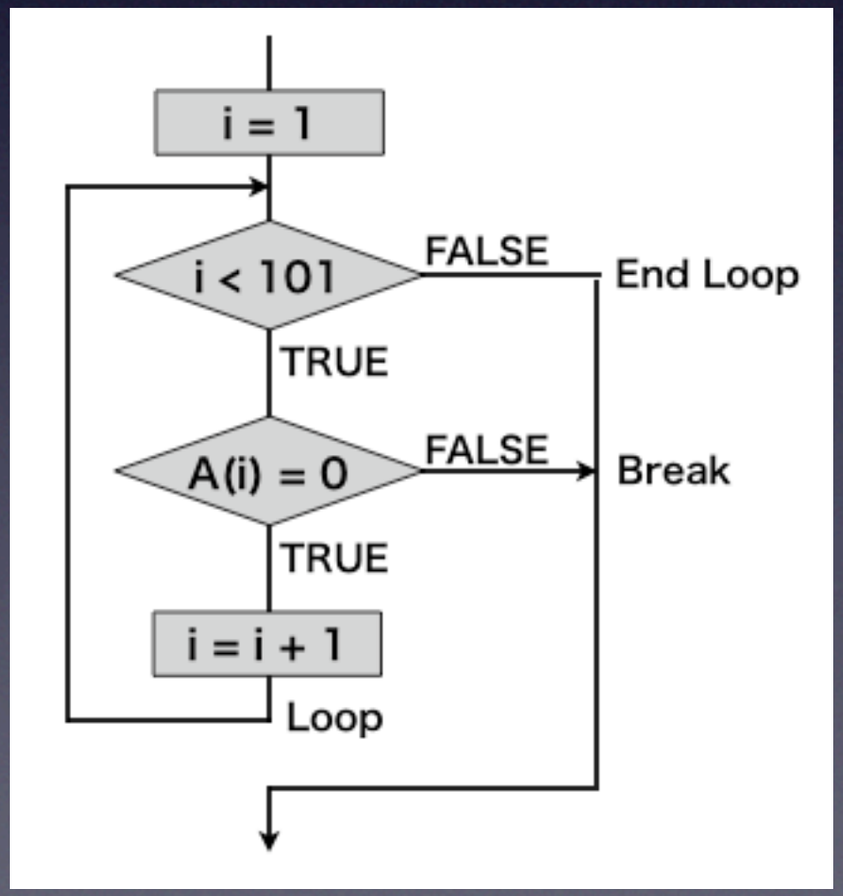
ループを繰り返すことにより、制御パスが爆発的に増加するという問題

↓

ループ回数に同値分割・境界値分析の考え方を適用し、実質的なパスを削減する

↓

複数の任意のループ回数を対象とした制御パスのテストケース作成が可能となる

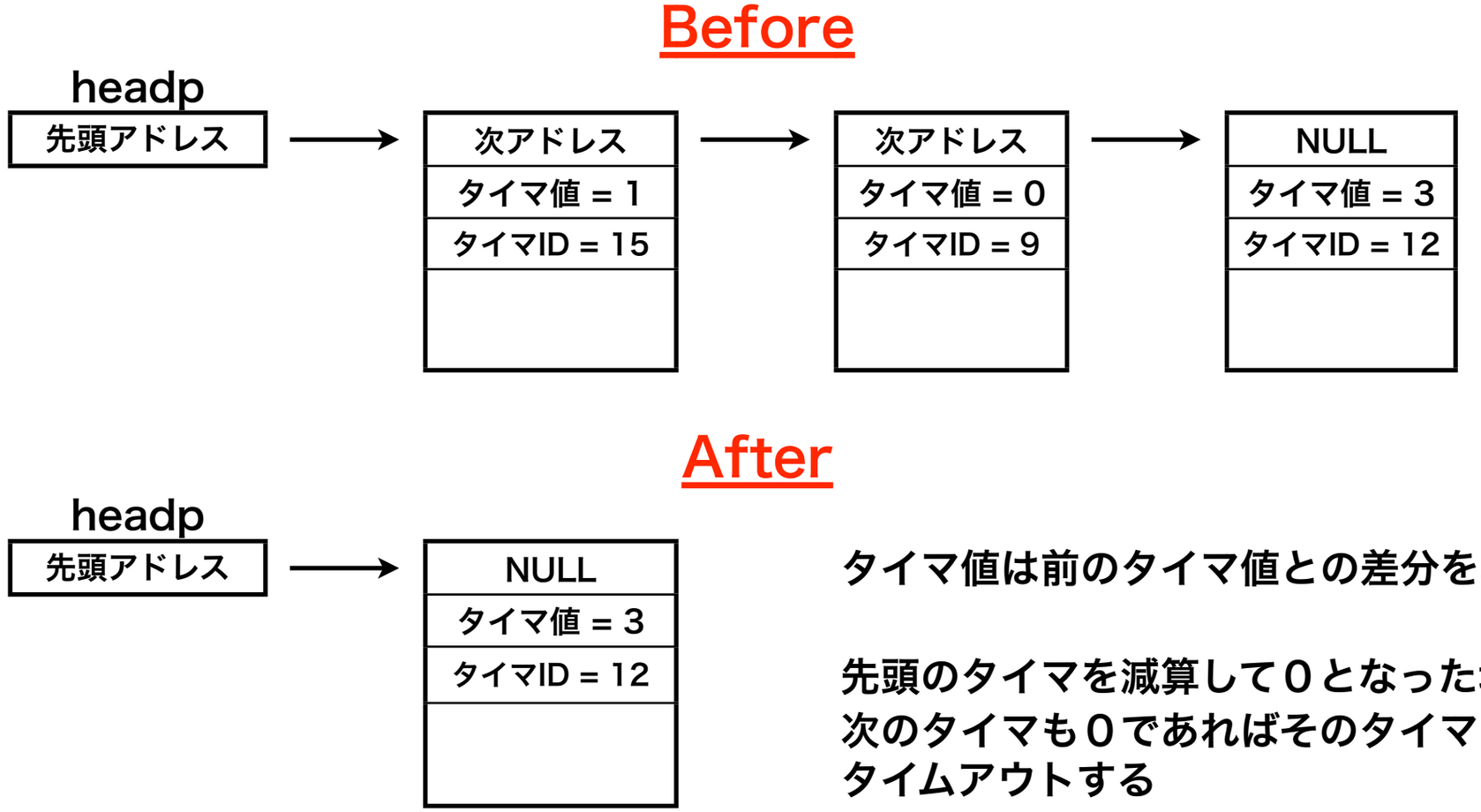


左のForループでループ回数が1回、2回 、100回
および100回を超えた場合のテストケース

No.	For	A(i)	Loop
1	i<101 TRUE	i=1 FALSE	Break
2	i<101 TRUE	i=100 FALSE	Break
3	i<101 TRUE	i=2 FALSE	Break
4	i<101 TRUE	TRUE	Loop
5	i=101 FALSE	-	End Loop

ループを含む関数の制御パステストへの適用例

タイマ更新関数の処理イメージ



タイマはメモリブロックとして確保され、キューイングされる
その際、タイマ値は既存タイマのタイマ値との差分に変更される
タイマ更新関数は先頭のタイマ値のみマイナス１すればよい

タイマ更新関数のソースリスト

```
 1 struct TMR {
 2     struct  TMR *nextp; //次タイマアドレス
 3     int     tmrct;      //タイマ値
 4     int     tmrid;      //タイマID
 5     int     tskid;      //タスクID
 6 }
 7
 8 struct TMR *headp;       //タイマ先頭アドレス
 9
10 int tmrcnt(void)         //タイマ減算処理
11 {
12     struct TMR *tp;      //タイマアドレス
13     int     cd;
14
15     cd = 0;
16     if (headp == NULL) {    //タイマが１つもない
17         return (0);
18     }
19     for (;;) {
20         if (headp->tmrct == 0) {    //タイマ値が0
21             tp = headp;
22             headp = headp->nextp;   //タイマをキューから外す
23             msg_snd(tp);    //タイムアウトメッセージ送信
24         } else {
25             if (cd == 1) {  //タイマ値が0のタイマから0以外のタイマへ
26                 return (1); //タイマ減算をやめる
27             }
28             headp->tmrcnt -= 1:     //タイマ値を1減算
29             if (headp->tmrcnt == 0) {   //タイマ値が0
30                 cd = 1;
31                 tp = headp;
32                 headp = headp->nextp;   //タイマをキューから外す
33                 msg_snd(tp);    //タイムアウトメッセージ送信
34             } else {
35                 return (0); //タイマ減算して0にならない
36             }
37         }
38         if (headp == NULL) {    //末尾のタイマ
39             return (cd);
40         }
41     }
42 }
```

制御パステストのモデルと制約表

パラメータはソースの行番号を表している　値の名称はそのパラメータで有効な条件の値を表す　B20はタイムアウトするタイマの数を３通りに分けている

制約表は制約対象が次の制約の制約条件になるという簡単なルールで記入する

パラメータ	値の並び
B16	NULL, NULL以外
for19	for, -
B20	タイマ値=0（1回）, タイマ値=0（2回）, タイマ値=0（3回）, タイマ値<>0, -
B25	タイマ値=0 → <>0, その他, -
B29	タイマ値=0, タイマ値<>0, -
B38	タイマは末尾, タイマは末尾でない, -
loop41	loop, -

制約表

パラメータ	制約１	制約２	制約３	制約４	制約５	制約６	制約７
B16	NULL	NULL以外					
for19	-	for	for				
B20	-		タイマ値=0（1回）, タイマ値=0（2回）, タイマ値=0（3回）, タイマ値<>0	タイマ値=0（1回）	タイマ値=0（2回）	タイマ値=0（3回）	タイマ値<>0
B25	-			-	タイマ値=0 → <>0, その他	タイマ値=0 → <>0, その他	タイマ値=0 → <>0, その他
B29	-			-			
B38	-			-			
loop41	-			-			

制御パステストの制約表（その２）

この制約表ではFor文でループする場合を制約13の「loop」で表している

制約８	制約９	制約１０	制約１１	制約１２	制約１３
タイマ値=0 → <>0	その他				
-	タイマ値=0, タイマ値<>0	タイマ値=0	タイマ値<>0		
-		タイマは末尾, タイマは末尾でない	-	タイマは末尾	タイマは末尾でない
-			-	-	loop

ソースリストを見ながら制約表へ記入できる

複数のループが入れ子になっている場合でも問題なく記述できる

制御パステストのテストケース

このテストケースではタイムアウトするタイマが１～３個の場合を対象に含めている
実際のテスト場面では、各テストケースの条件に合うようにメモリの書き換えを行ない、関数を実行する

No.	B16	for19	B20	B25	B29	B38	loop41
1	NULL	-	-	-	-	-	-
2	NULL以外	for	タイマ値<>0	タイマ値=0 → <>0	-	-	-
3	NULL以外	for	タイマ値<>0	その他	タイマ値=0	タイマは末尾でない	loop
4	NULL以外	for	タイマ値<>0	その他	タイマ値<>0	-	-
5	NULL以外	for	タイマ値<>0	その他	タイマ値=0	タイマは末尾	-
6	NULL以外	for	タイマ値=0（1回）	-	-	-	-
7	NULL以外	for	タイマ値=0（2回）	タイマ値=0 → <>0	-	-	-
8	NULL以外	for	タイマ値=0（2回）	その他	タイマ値=0	タイマは末尾	-
9	NULL以外	for	タイマ値=0（2回）	その他	タイマ値=0	タイマは末尾でない	loop
10	NULL以外	for	タイマ値=0（2回）	その他	タイマ値<>0	-	-
11	NULL以外	for	タイマ値=0（3回）	その他	タイマ値=0	タイマは末尾でない	loop
12	NULL以外	for	タイマ値=0（3回）	タイマ値=0 → <>0	-	-	-
13	NULL以外	for	タイマ値=0（3回）	その他	タイマ値<>0	-	-
14	NULL以外	for	タイマ値=0（3回）	その他	タイマ値=0	タイマは末尾	-

制御パステストのテストケースをPairwise法で作成できる

その制約指定の手段として制約表の使用が有効である

Pairwise法による制御パステストの性質

制御パステストと組み合わせテストの対比

制御パステストの分類	カバレッジによる分類	対応する組み合わせテスト
命令網羅	最も低い	(要因限定テスト)
条件網羅	命令網羅より高い	(要因列挙テスト)
条件網羅(Pairwise法)	条件網羅より高い	全ペア組み合わせテスト
経路網羅	最も高い	全数組み合わせテスト

Pairwaise法による制御パステストの特長

- ツールを使うことで簡単にテストケースが作れる
- どのような値を選択するかによってテストの粒度を調節できる
- 複数の要因の組み合わせによって発生するバグを発見できる

Pairwaise法による制御パステストが向いている対象

- 特に重要で重点的にテストしたいコンポーネント

組み合わせテスト実施上の留意点

- 組み合わせテストはテストケース数が多くなりがち
  適用する対象を限定する
- ソフト開発者の視点も取り入れてテスト設計を行なう
- テスト仕様書の版数管理を行なう
- 組み合わせるパラメータと値の選択が最も重要

ご静聴ありがとうございました

岩通ソフトシステム株式会社

JaSST 2009 東京