単体テスト支援ツール「パス解析システム」

― パスに基づく真の品質管理へ向けて ―

オーエスエー・リミテッド

新原　俊一

会社名：オーエスエー・リミテッド合同会社

URL ：ＯＳＡ－ltd. Com

所在地： 熊本県阿蘇郡南阿蘇村

設　立 ：　2007年4月

目　的 ：・コンピュータシステムの研究開発

・ライセンス販売

・コンサルティング　　他

目　次

単体テストの基本：プログラムはパスに沿って実行される

「基本から学ぶソフトウェアテスト」テスト技術者交流会（ＴＥＦ）訳　より引用

．．．全パスを網羅しない限り、完全なテストをしたと言えない・・・

テストを省略しても良いと思えるパスが有るかもしれないが、そのパスにはそのパス特有の問題が有るものだ。

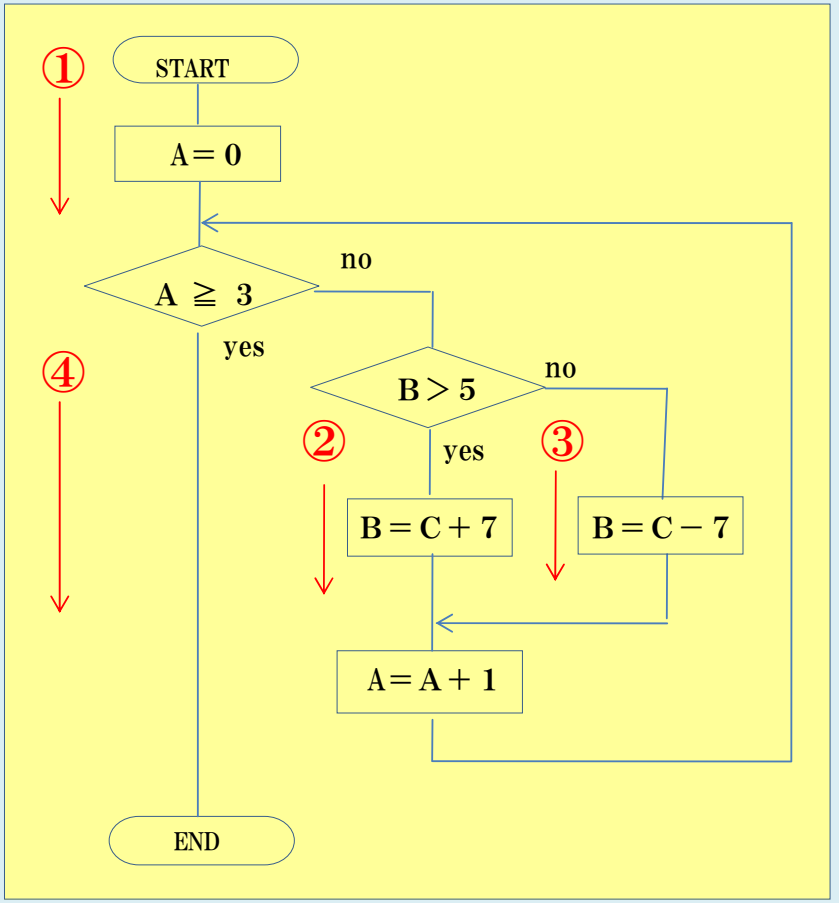
プログラムはパスに沿って実行される

↓

パステストは基本中の基本

パステストの現状と課題：パステストの大部分は分岐のパスのテスト

では、パステストの現状はどうなっているのでしょうか



我々が通常行っているパステストの大部分は、

①→②→④
①→③→④
①→④

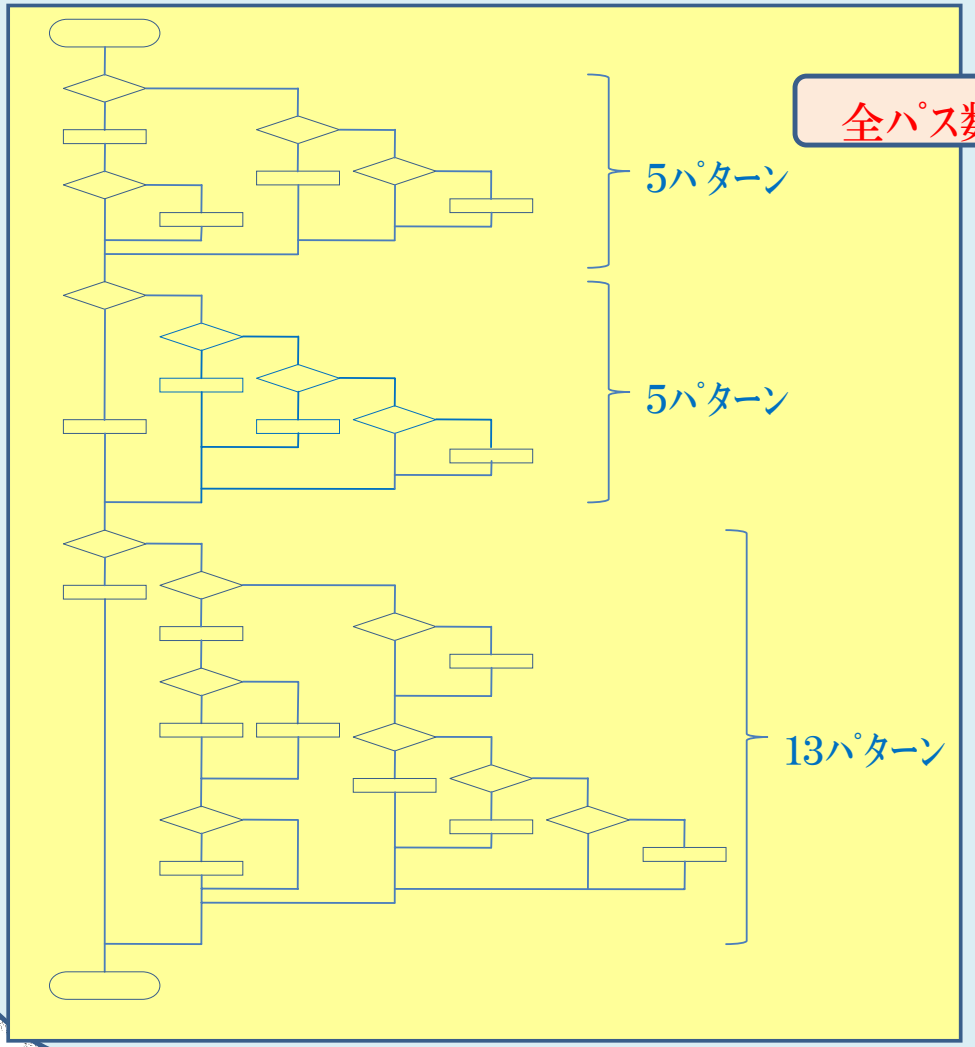
の分岐のパステストです。

①→②→②→②→④、①→②→②→③→④
①→②→③→②→④、①→②→③→③→④
①→③→②→②→④、①→③→②→③→④
①→③→③→②→④、①→③→③→③→④

といった繰返しのパスは、ほとんどテストされていません。

パステストの現状と課題：分岐のパスの一部しかテストしていない

その分岐のパスの全てがテストされているでしょうか



全パス数　＝　5×5×13＝325 パターン

我々は、通常どの程度テストケースを作成しているでしょうか？

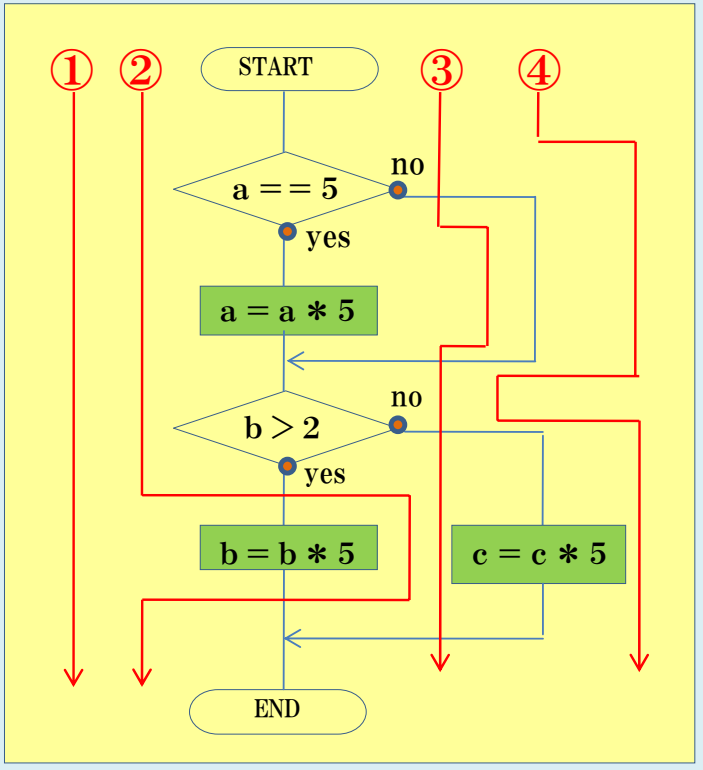
50？　70？　100？

こんものではないでしょうか。

つまり、分岐のパスもその一部しかテストしていないのです。

パステストの現状と課題： 不透明な品質測定

パステストの品質測定方式も分岐のパスが対象です



$$C_0\text{メジャー} = \frac{\text{最低一度実行した命令数}}{\text{全命令数}}$$

①、②を実行すれば網羅率100％

③、④は実行されなくてもよい

$$C_1\text{メジャー} = \frac{\text{最低一度実行した分岐数}}{\text{全分岐数}}$$

①、④を実行すれば網羅率100％

②、③は実行されなくてもよい

全パスの中、どの程度をテストしたか不透明

パステストの現状と課題： 続発するコンピュータシステムの事故

このような状況の下、コンピュータシステムの事故は絶えることが有りません

- **全国の社会保険事務所で年金システムダウン**

  オンライン・システムを制御するミドルウエアの不具合「2007年　日経コンピュータ」

  全国23県合計130の社会保険事務所で端末が起動せず、年金の照会業務ができなかった。

- **首都圏でSuicaの大規模トラブル、改札ゲートが閉鎖**

  Suicaの状態をチェックするプログラムにミス「2006年　日経コンピュータ」

  首都圏3分の1以上の駅で改札ゲートが閉鎖。

- **航空管制システムで障害、30万人以上が足止め**

  システム改修時のバグ「2003年　日経コンピュータ」

  航空管制システム障害は、欠航215便、大幅な遅延1500便以上、足止めされた客30万人以上と、

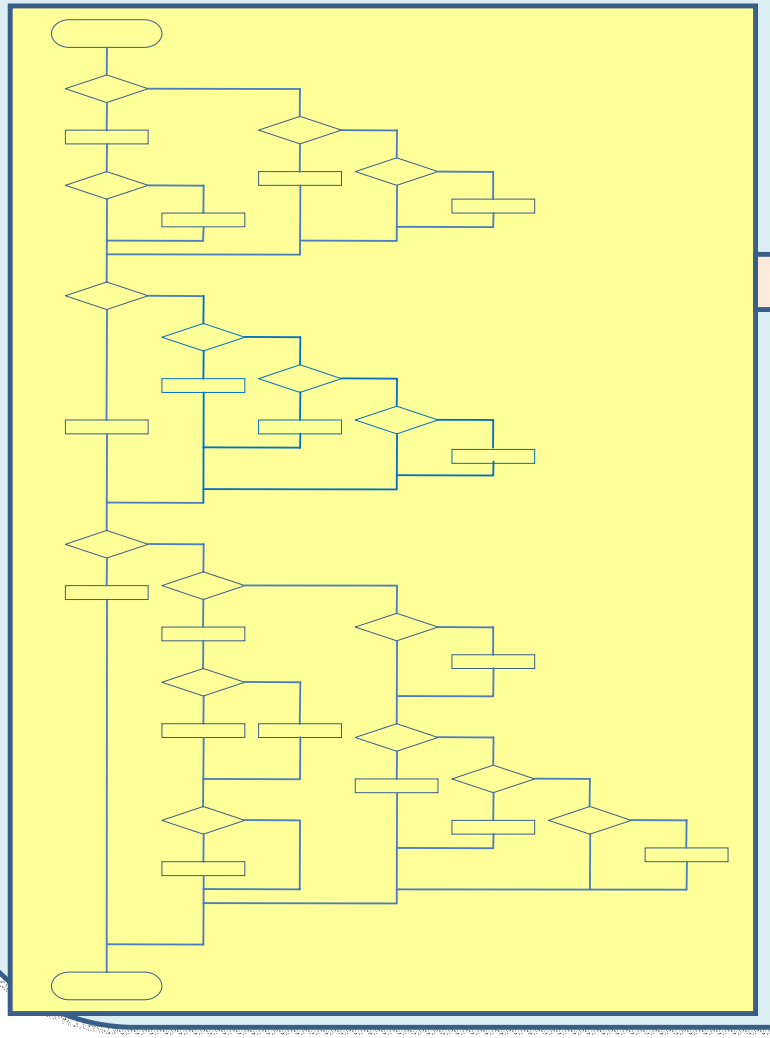
- **「ディスカバリー」の不正プログラミング**「1990　NASA」

  エラーはスペースシャトル搭乗員によってミッション開始後1時間後に発見され、直ちに訂正された。

  （「危ないコンピューター頻発するコンピュータ事故からの教訓－」 ピータ・ニューマン著）

パステストの現状と課題： パスの自動抽出技術が不可欠

何故、プログラム不良による事故が続発しているのに分岐のパスの一部しか
テストされていないのでしょうか



現在、パスを抽出するには人手で抽出するしかありません。
フローチャートの作成には時間がかかるうえミスが多発します。

ビジネス効率が悪すぎる

－　真のパステストの課題　－

迅速かつ正確に全パスを抽出する自動化技術を開発すること

条件分岐パターン自動解析技術（特許出願中）

条件分岐パターン自動解析技術は、分岐処理系のパスを自動で抽出する技術

No	プログラムソース	次処理ステップ 真	次処理ステップ 偽
1	IF (a == 5)	2	7
2	{　a = a * b;	3	
3	IF (c > 2)	4	6
4	{　c = c * d;	5	
5	}	6	
6	}	13	
7	ELSE	8	
8	{	9	
9	IF (e < 5)	10	12
10	{　e = e * f;	11	
11	}	12	
12	}	13	
13	a = c * e;		

この技術は３つの機能で構成される

＜Ｎｏ１．ＩＦ文節の認識＞

各ＩＦ文節を正しく認識し、制御の流れを把握する必要があります。

文法とスタックの技術を応用しネスト構造を解析し実現しています。

＜Ｎｏ２．処理順の割付け＞

ソースの各ステップに、次に処理するステップNoを割付けます。

但し、ＩＦ文では、真の条件と偽の条件の両方を割付ける。

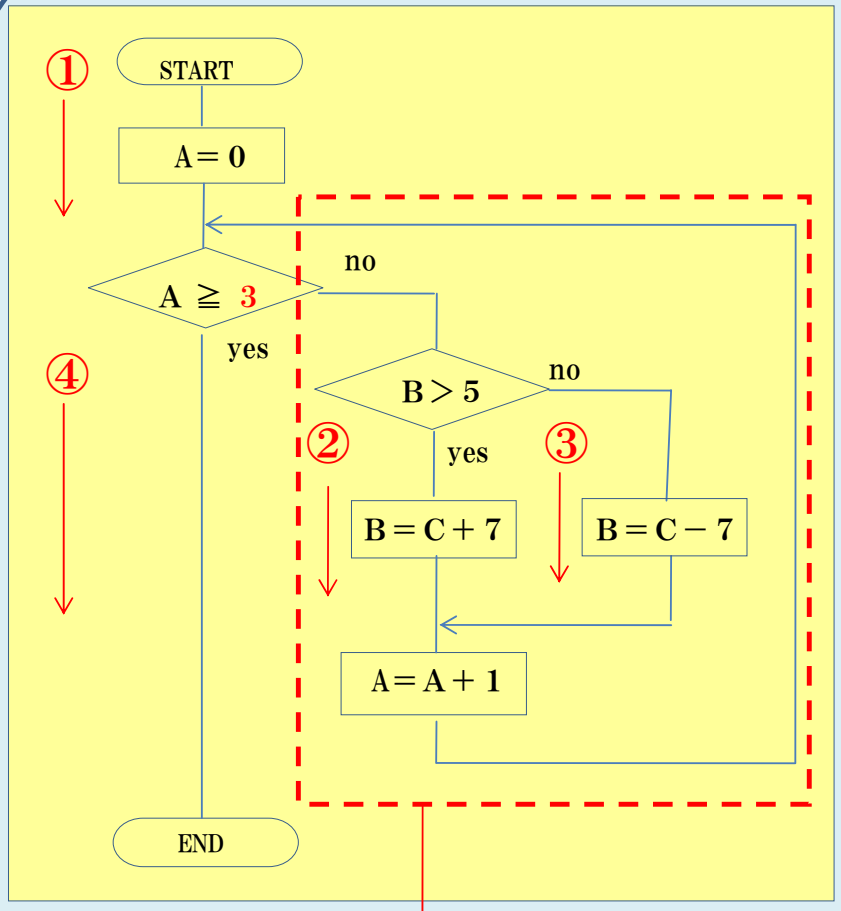
条件分岐パターン自動解析技術（特許出願中）

＜Ｎｏ３．パスの抽出＞　独自の探索技術でパスを抽出します

No	プログラムソース	次処理ステップ 真	次処理ステップ 偽	(1) ①	(2)	(3) ②	(4)	(5) ③	(6)	(7) ④	(8)
1	IF (a == 5)	2	7	1		1	7	1		1	
2	{　a = a * b;	3		2		2					
3	IF (c > 2)	4	6	3	6	3					
4	{　c = c * d;	5		4							
5	}	6		5							
6	}	13		6		6					
7	ELSE	8						7		7	
8	{	9						8		8	
9	IF (e < 5)	10	12					9	12	9	
10	{　e = e * f;	11						10			
11	}	12						11			
12	}	13						12		12	
13	a = c * e;			13		13		13		13	

繰返し処理系パス自動解析技術：固定型

繰返し処理系パス自動解析技術は、繰返し処理系のパスを自動で抽出する技術



繰返し処理系
- 固定型
- 不定型

＜Ｎｏ１．固定型の抽出技術＞

繰返し処理系の繰返し回数が固定のタイプ。（例：配列の初期化処理）

下段の式に従ってパスを抽出。

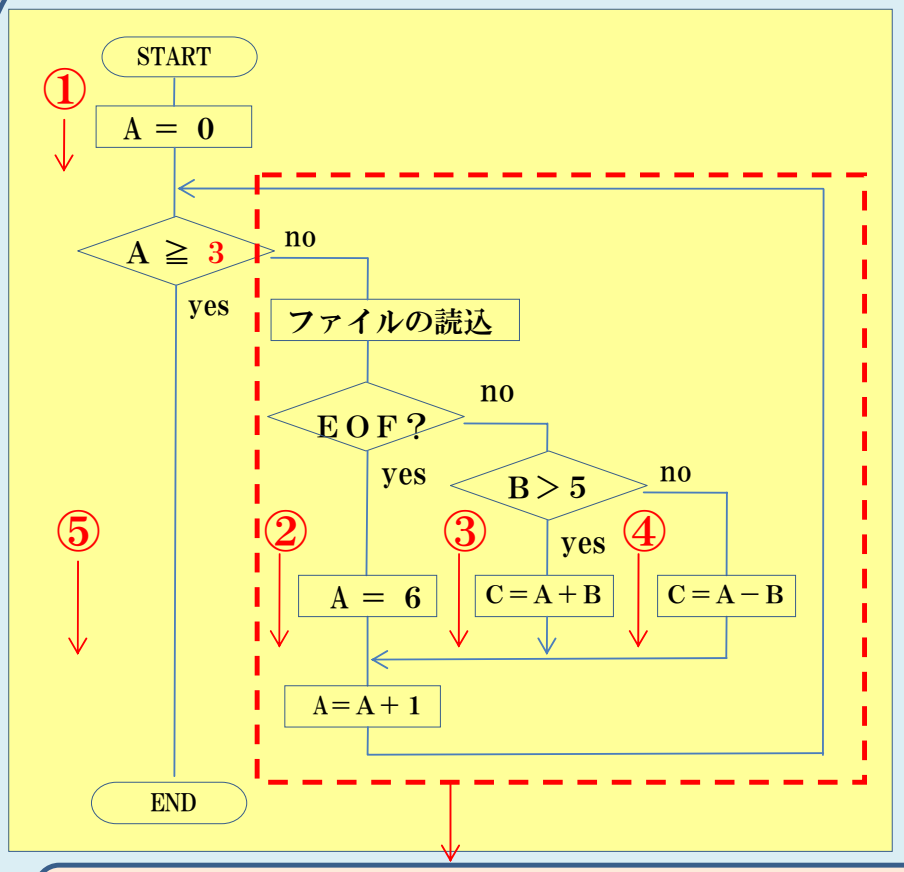
（例：・分岐のパス数　：２

・最大繰返し回数：３

全パス数＝2^3　＝８パターン）

繰返し処理系内の全パス数　＝　分岐のパス数の“最大繰返し回数”乗

繰返し処理系パス自動解析技術：不定型



＜Ｎｏ２．不定型の抽出技術＞

繰返し処理系の最大繰返し回数の中で繰返し回数が不定（実行時に定まる）のタイプ。

下段の式に従ってパスを抽出。

（例：・分岐のパス数　：３

・最大繰返し回数：3_1

全パス数＝繰返し回数1 ⇒３

＋ 繰返し回数２⇒ 3^2

＋ 繰返し回数３⇒ 3^3

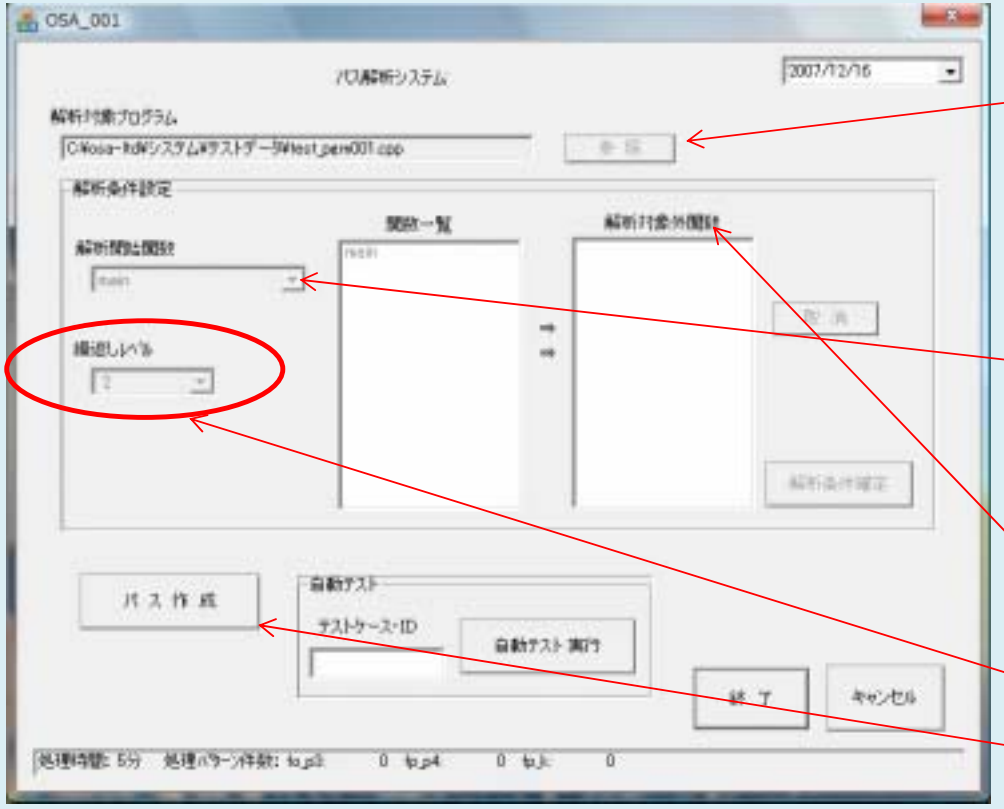
＝３９ﾊﾟﾀｰﾝ）

繰返し処理系内の全パス数 ＝ 分岐のパス数の１乗 ＋ 分岐のパス数の２乗 ～

＋ 分岐のパス数の“最大繰返し回数”乗

「パス解析システム」プロトタイプ：概　要

パスの一覧表を作成し、デバッグ時の人的ミスを減らすことを目的



（1）解析対象プログラムの登録
：コンパイルが完了したＣ言語のプログラムソースを選択
・ＧＯＴＯ文不可　他

（2）解析開始関数の登録
：Ｃ言語では通常ｍａｉｎ関数ですが、複雑な特定の関数を解析する場合に指定

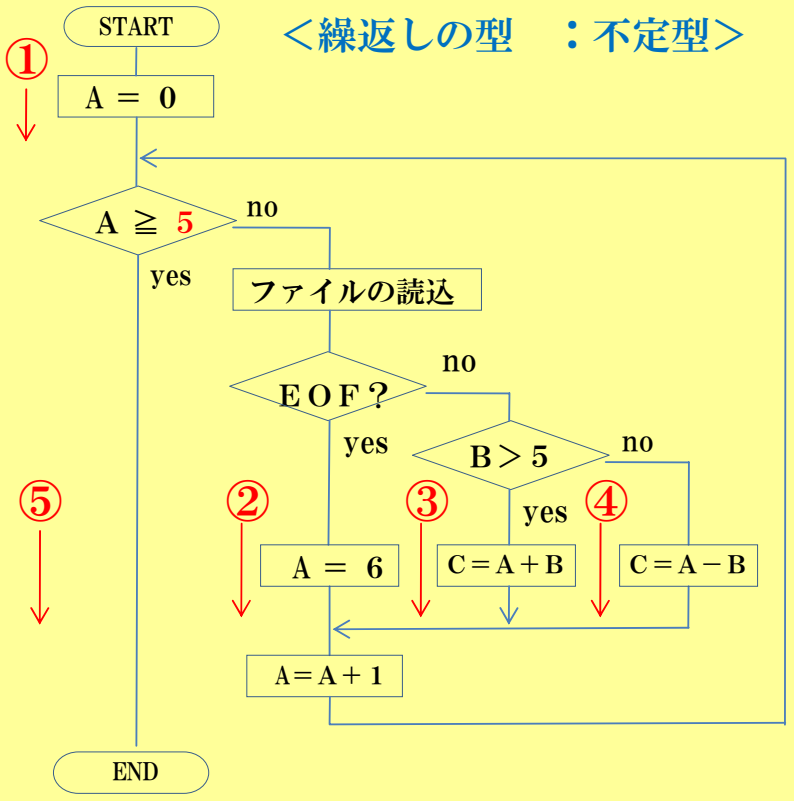
（3）解析対象外関数の登録
：指定の関数のパスを作成しない機能

（4）繰返しレベルの登録

（5）「処理手順一覧表」作成

「パス解析システム」プロトタイプ：概　要

（４）繰返しレベルの登録：ソース中の繰返し回数そのままでは莫大なパス数となるため指定の回数に置き換えパスを削減する機能



・ソース中の繰返し回数　：５回

$$\overset{1}{1}+\overset{2}{3}+\overset{3}{3}+\overset{4}{3}+\overset{5}{3}+3 \quad =364\ パターン$$

・繰返しレベル：1

$$\overset{1}{1}+3 \quad =4\ パターン$$

・繰返しレベル：2

$$\overset{1}{1}+\overset{2}{3}+3 \quad =13\ パターン$$

・繰返しレベル：3

$$\overset{1}{1}+\overset{2}{3}+\overset{3}{3}+3=40\ パターン$$

「パス解析システム」プロトタイプ：概　要

（５）「処理手順一覧表」作成

プログラムソースから抽出したパスの内容を処理順に並べたもの。

識別子には条件が真の場合は‘Ｔ’、偽の場合は‘Ｅ’が設定されており条件を確認しながらシンプルに処理を辿ることができるため、人的ミスを減らすことができます。

```
IF (a == 5)
{   a = a * b;
    IF (c > 2)
    {  c = c * d;
    }
}
ELSE
{
    IF (e < 5)
    {  e = e * f;
    }
}
a = c * e;
```

プログラムソース

パスＮｏ：③

識別子	内　容
	IF (a == 5)
E	{
	IF (e < 5)
T	{ e = e * f;
	}
	}
	a = c * e;

「処理手順一覧表」

「パス解析システム」プロトタイプ：適用事例

＜適用事例＞・ステップ数　：２８ｽﾃｯﾌﾟ・繰返しの型：不定型
・繰返しレベル：２

```
void if_chk()
{  for(idx_pgm21=0,end_pgm1=0;end_pgm1 == 0;idx_pgm21++)
  { for(idx_pgm22=0,end_pgm2=0;end_pgm2 == 0;idx_pgm22++)
    { switch (tbl_pgm2[idx_pgm21].bun[idx_pgm22])
      { case EOF:
          end_pgm1 = 1;
          end_pgm2 = 1;
          break;
        case '¥n':
          end_pgm2 = 1;
          break;
        case 'i':
          if (tbl_pgm2[idx_pgm21].bun[idx_pgm22 + 1] == 'f')
          { if ((tbl_pgm2[idx_pgm21].bun[idx_pgm22 + 2] == ' ') ||
                (tbl_pgm2[idx_pgm21].bun[idx_pgm22 + 2] == '('))
            { if (tbl_pgm2[idx_pgm21 + 1].bun[0] != '{')
              { err_msg = 11;
                end_pgm1 = 1;end_pgm2 = 1;
              }
            }
          }
          break;
        default:
          break;
      }
    }
  }
}
```

2 パターン

7 パターン

＜パスの求め方＞

① 第一繰返しをやらない　1

② 第一繰返しをやる
- 2^1 ×　1
- 2^1 ○　56
- 2^2 ○→○　56×56＝3136
- 2^2 ○→×　56×1＝56
- 2^2 ×→○　1×56＝56
- 2^2 ×→×　1×1＝1

合計　3，307ﾊﾟﾀｰﾝ

③ ×　:第二繰返しをやらない　1 ﾊﾟﾀｰﾝ

④ ○　:第二繰返しをやる　$7^1 + 7^2 = 56$ ﾊﾟﾀｰﾝ

結論

３，３０７ﾊﾟﾀｰﾝもチェックできない！

「パス解析システム」プロトタイプ：考　察

＜考　察＞

この膨大なパス数を前にした時、

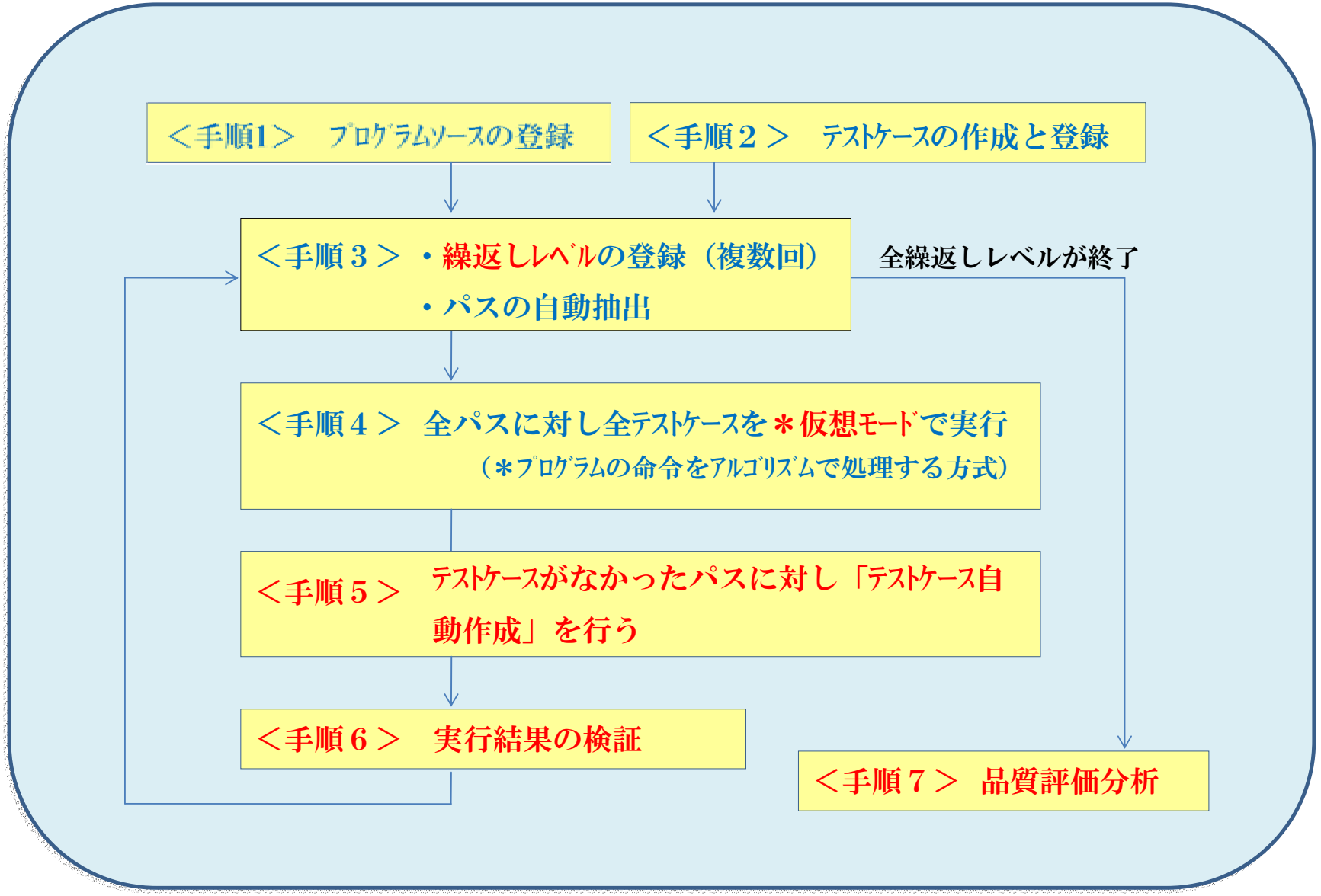
当方式では無理にパス数を削減することになり、単体テストの実態と乖離した結果になる

「パス解析システム」の開発中止

単体テストの実態である膨大なパス数を前提とした方式を検討

「自動テスト方式」の研究開始

自動テスト方式：概　要


＜手順1＞　プログラムソースの登録
＜手順２＞　テストケースの作成と登録
＜手順３＞　・繰返しレベルの登録（複数回）
・パスの自動抽出
全繰返しレベルが終了
＜手順４＞　全パスに対し全テストケースを＊仮想モードで実行
（＊プログラムの命令をアルゴリズムで処理する方式）
＜手順５＞　テストケースがなかったパスに対し「テストケース自動作成」を行う
＜手順６＞　実行結果の検証
＜手順７＞　品質評価分析

自動テスト方式：＜手順５＞「ﾃｽﾄｹｰｽ自動作成」

テストケースが無かったパスからテストケースを自動作成します

＜step １＞　ﾃｽﾄｹｰｽが無かったパスの抽出

全ﾊﾟｽが終了

E　N　D

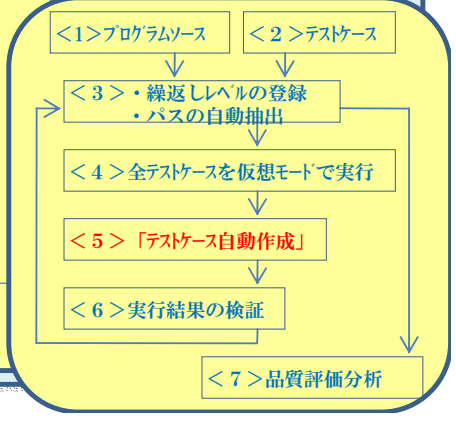
＜ step ２＞　初期値が与えられていない変数への
初期値の割当

＜ step ３＞

① トレースを取得しながら仮想ﾓｰﾄﾞで実行

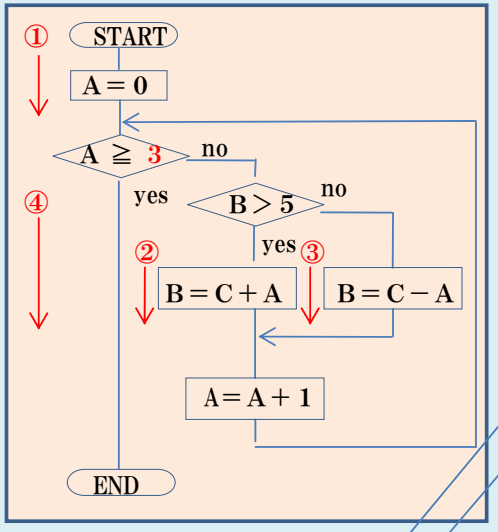
② 仮想ﾓｰﾄﾞとパスの条件判定が不一致の場合

- パスの条件判定に一致するよう変数の値を求める
- トレースに従いパスの先頭まで逆処理を行う。
  但し、パスが無効パスの場合は処理を終了する。
- 変数への初期値の再設定後①へ



「テストケース自動作成」 ①トレースを取得しながら仮想モードで実行

＜ フローチャート ＞



＜ パス ＞

	No	条件判定	処理内容
			（ＳＴＡＲＴ）
①	1		Ａ＝０
	2	Ｎｏ	Ａ≧３
	3	Ｙｅｓ	Ｂ＞５
②	4		Ｂ＝Ｃ＋Ａ
	5		Ａ＝Ａ＋１
	6	Ｎｏ	Ａ≧３
	7	Ｙｅｓ	Ｂ＞５
②	8		Ｂ＝Ｃ＋Ａ
	9		Ａ＝Ａ＋１
	１０	Ｎｏ	Ａ≧３
	１１	Ｙｅｓ	Ｂ＞５
②	１２		Ｂ＝Ｃ＋Ａ
	１３		Ａ＝Ａ＋１
④	１４	Ｙｅｓ	Ａ≧３
			（ＥＮＤ）

（１）トレースを取得しながら仮想モードでの実行１回目

トレース　初期値設定：Ａ＝０、Ｂ＝６、Ｃ＝２

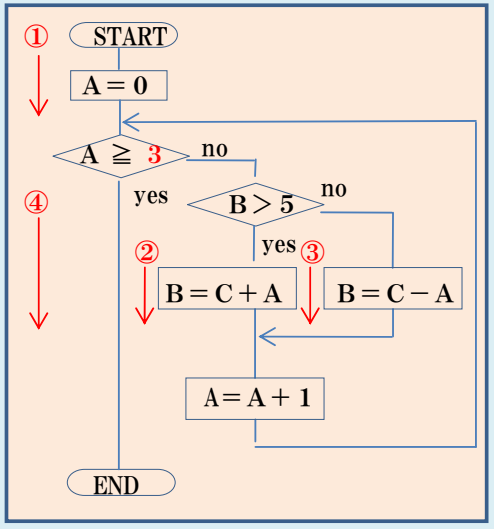
１	処理	Ａ＝０				
２	条件判定	Ａ≧３	→	０≧３	⇒	Ｎｏ （パスと一致）
３	条件判定	Ｂ＞５	→	６＞５	⇒	Ｙｅｓ（パスと一致）
４	処理	Ｂ＝Ｃ＋Ａ	→	２＋０	→	Ｂ＝２
５	処理	Ａ＝Ａ＋１	→	０＋１	→	Ａ＝１
６	条件判定	Ａ≧３	→	１≧３	⇒	Ｎｏ （パスと一致）
７	条件判定	Ｂ＞５	→	９＞５	⇒	Ｙｅｓ（パスと一致）
８	処理	Ｂ＝Ｃ＋Ａ	→	２＋１	→	Ｂ＝３
９	処理	Ａ＝Ａ＋１	→	１＋１	→	Ａ＝２
１０	条件判定	Ａ≧３	→	２≧３	⇒	Ｎｏ （パスと一致）
１１	条件判定	Ｂ＞５	→	９＞５	⇒	Ｙｅｓ（パスと一致）
１２	処理	Ｂ＝Ｃ＋Ａ	→	２＋２	→	Ｂ＝４
１３	処理	Ａ＝Ａ＋１	→	２＋１	→	Ａ＝３
１４	条件判定	Ａ≧３	→	３≧３	⇒	Ｙｅｓ（パスと一致）

・テストケース：ＩＮＰＵＴ　Ａ＝０、Ｂ＝６、Ｃ＝２

ＯＵＴＰＵＴ　Ａ＝３、Ｂ＝４、Ｃ＝２

「テストケース自動作成」 ② 仮想モードとパスの条件判定が不一致

＜ フローチャート ＞



＜ パス ＞

	No	条件判定	処理内容
①			（ＳＴＡＲＴ）
	1		Ａ＝０
②	2	Ｎｏ	Ａ≧３
	3	Ｙｅｓ	Ｂ＞５
	4		Ｂ＝Ｃ＋Ａ
	5		Ａ＝Ａ＋１
③	6	Ｎｏ	Ａ≧３
	7	Ｎｏ	Ｂ＞５
	8		Ｂ＝Ｃ－Ａ
	9		Ａ＝Ａ＋１
②	１０	Ｎｏ	Ａ≧３
	１１	Ｙｅｓ	Ｂ＞５
	１２		Ｂ＝Ｃ＋Ａ
	１３		Ａ＝Ａ＋１
④	１４	Ｙｅｓ	Ａ≧３
			（ＥＮＤ）

（１）トレースを取得しながら仮想モードでの実行１回目

トレース　初期値：Ａ＝０、Ｂ＝６、Ｃ＝７

１	処理	Ａ＝０	
２	条件判定	Ａ≧３ → ０≧３ ⇒ Ｎｏ	（パスと一致）
３	条件判定	Ｂ＞５ → ６＞５ ⇒ Ｙｅｓ	（パスと一致）
４	処理	Ｂ＝Ｃ＋Ａ→７＋０→Ｂ＝７	
５	処理	Ａ＝Ａ＋１→０＋１→Ａ＝１	
６	条件判定	Ａ≧３ → １≧３ ⇒ Ｎｏ	（パスと一致）
７	条件判定	Ｂ＞５ → ７＞５ ⇒ Ｙｅｓ	（パスと不一致）

（２）条件に一致する値を境界値で計算

条件計算　Ｂ＞５ → Ｎｏ → Ｂ≦５ → Ｂ＝５

（３）トレースの中から条件を変更した変数（Ｂ）へ代入している処理を抽出し逆処理を行う

トレース　変数の値：Ａ＝１、Ｂ＝５、Ｃ＝７

４　処理　Ｂ＝Ｃ＋Ａ → Ｃ＝Ｂ－Ａ → ５－１ → Ｃ＝４

（４）トレースを取得しながら仮想モードでの実行２回目

トレース　初期値再設定：Ａ＝０、Ｂ＝５、Ｃ＝４

１	処理	Ａ＝０	
２	条件判定	Ａ≧３ → ０≧３ ⇒ Ｎｏ	（パスと一致）
３	条件判定	Ｂ＞５ → ５＞５ ⇒ Ｎｏ	（パスと不一致）

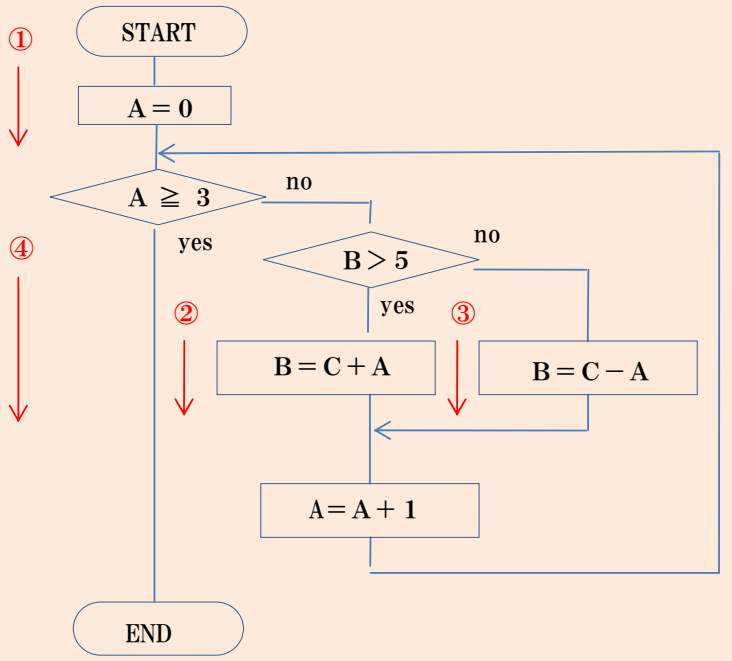
（５）条件に一致する値を境界値で計算

条件計算　Ｂ＞５ → Ｙｅｓ → Ｂ＝６

この後は、（１）～（５）のパターンを繰返す為、
①→②→③→②→④のパスは無効パスとなります

「テストケース自動作成」 有効パスと無効パス

＜　フローチャート　＞



＜テストケース＞

（1）

	A	B
①	0	6
④	−	−

無効パス

（2）

	A	B
①	0	6
②	0	6
②	1	6
②	26	
④	3	6

有効パス

（3）

	A	B
①	0	6
②	0	6
②	1	6
③	−	−
④		

無効パス

（4）

	A	B
①	0	6
②	1	6
③	−	−
②		
④		

無効パス

（5）

	A	B
①	0	6
②	16	
③	−	−
③		
④		

無効パス

（6）

	A	B
①	0	4
③	0	4
③	1	4
③	24	
④	3	4

有効パス

（7）

	A	B
①	0	4
③	0	4
③	1	4
②	−	−
④		

無効パス

（8）

	A	B
①	0	4
③	0	4
②	−	−
③		
④		

無効パス

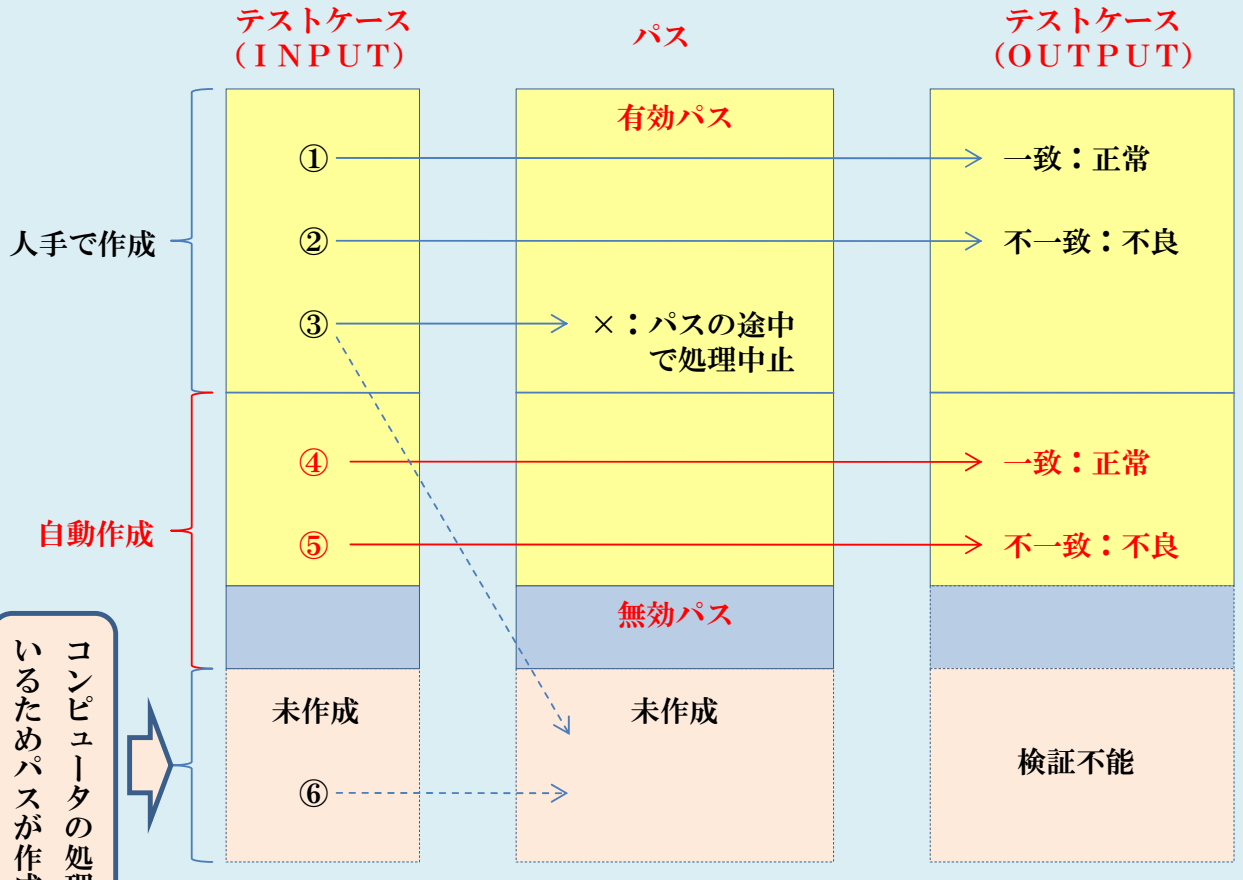
（9）

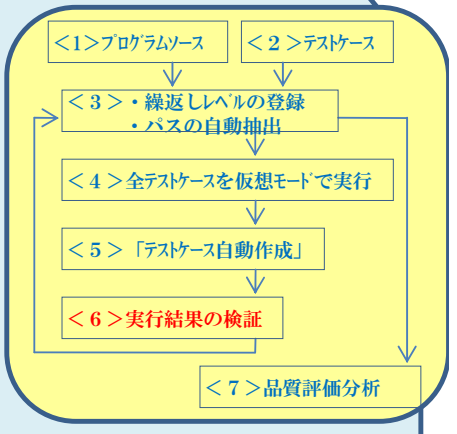
	A	B
①	0	4
③	0	4
②	−	−
②		
④		

無効パス

有効パス：2　　無効パス：7

自動テスト方式　＜手順６＞　実行結果の検証





コンピュータの処理能力を超えているためパスが作成できない領域

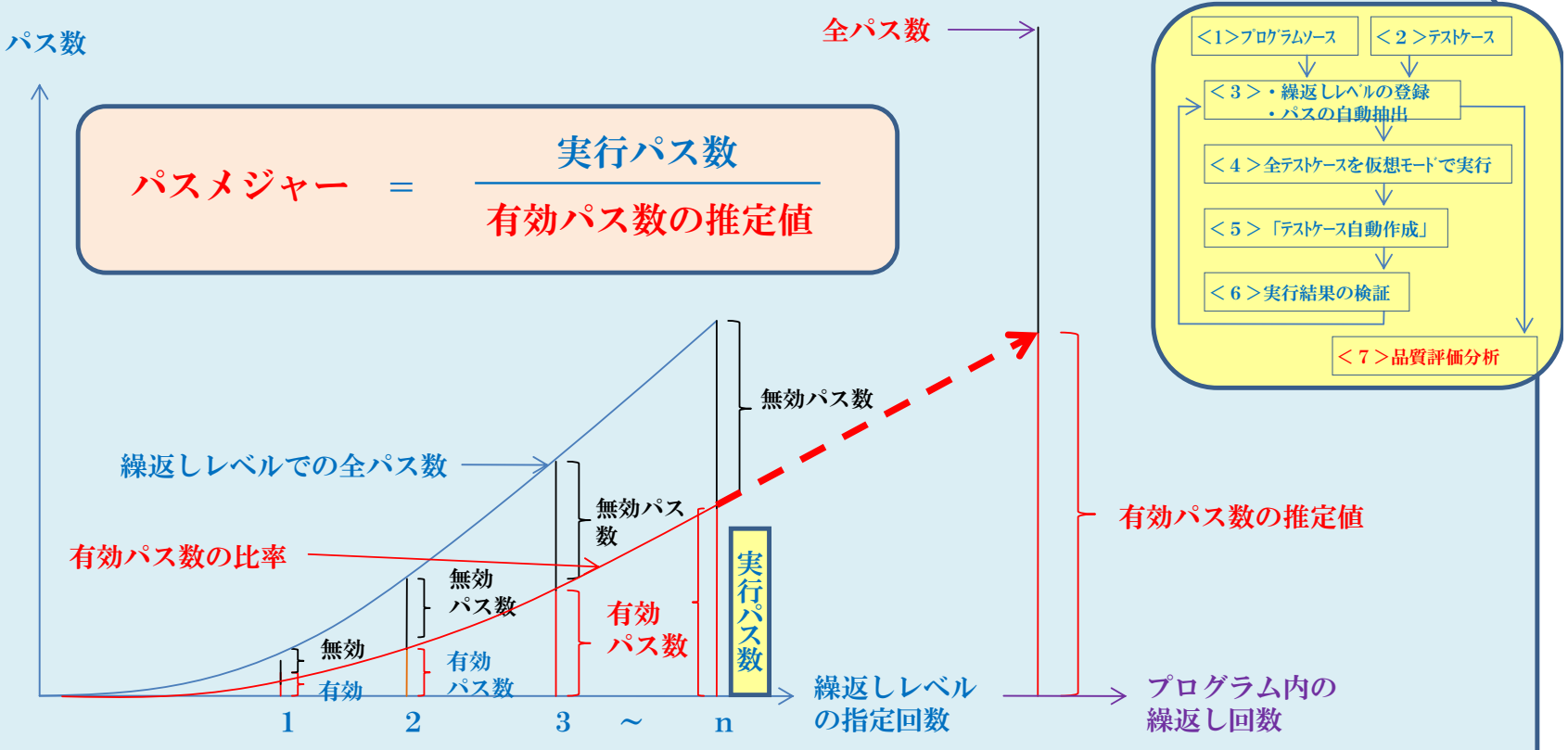
自動テスト方式により期待される効果

（１）人手で作成できなかったテストケースの漏れを抽出（④、⑤）

（２）人手で発見できなかったプログラム不良を摘出（⑤）

（３）実行環境が不要な為、環境構築等他の作業の影響を受けない

自動テスト方式　＜手順７＞品質評価分析：パスメジャーを求める



「有効パス数の推定値」の求め方

：プログラムの繰返し回数で求める有効パス数の多くはコンピュータの処理能力を超え求めることができません。従って、各繰返しレベルでの全パス数に占める有効パス数の比率を求め、それらから全パス数に占める比率を推定（傾向分析値）する。

有効パス数の推定値　＝　全パス数　×　有効パス比率の傾向分析値

不透明な管理から定量的管理へ

単体テストの現状　・全分岐のパスの一部しかテストができていない
・全パスの中、どの程度をテストしたか不透明

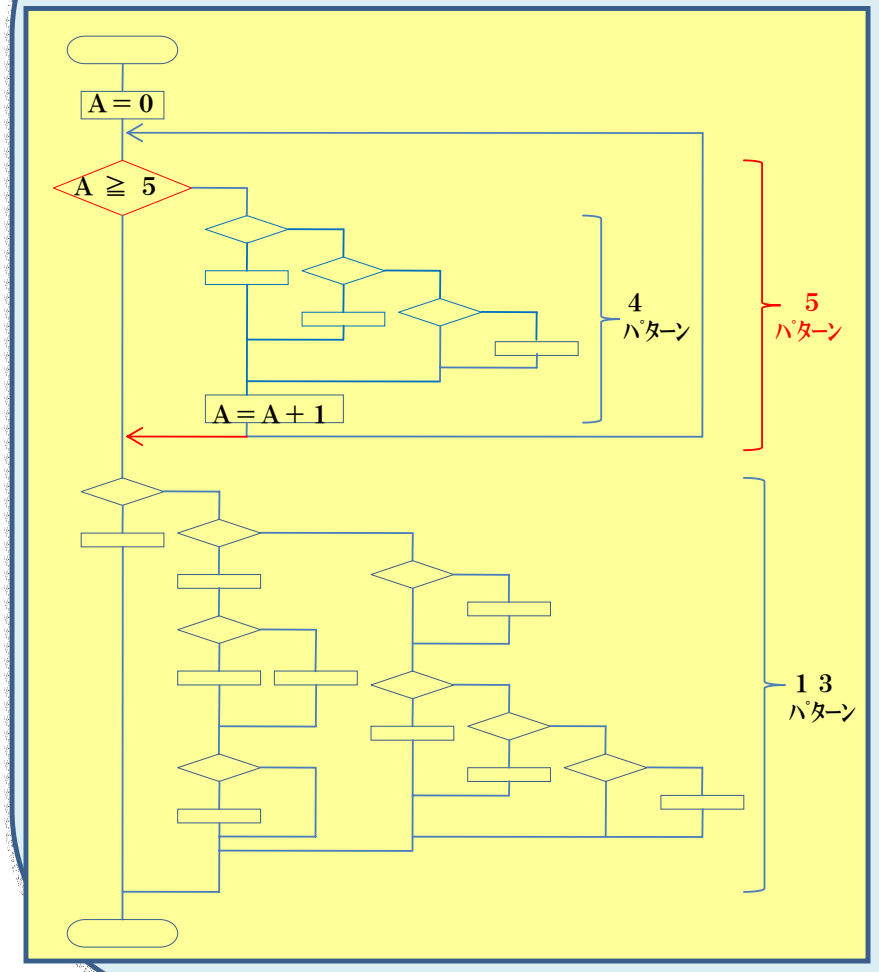
「自動テスト方式」により大幅なプログラム品質向上が期待できます

定量的管理が可能

第一段階：・全分岐のパスを対象とするテスト方式の導入
この方式の利点は、現在のテスト方式の延長であり、全分岐のパス数も実用的な範囲に収まる為導入しやすいことです。
・定量的管理：簡易パスメジャー

不透明な管理から定量的管理へ：第一段階 全分岐パスの定量的管理

$$\text{簡易パスメジャー} = \frac{\text{実行パス数（N）}}{\text{全分岐の有効パス数（M）}}$$



- 繰返しの型：固定型
- 最大繰返し回数：５回
- 繰返し処理内のパス数：４
- 全パス数＝（１＋4^{5}）×１３＝１３、３２５パターン

- 繰返しレベル：１
（繰返し命令が分岐命令に変換された状態になる）
- 全分岐のパス数＝５×１３＝６５パターン
- 全分岐の有効パス数　M ＝ ４０
- 実行パス数　N ＝ ３０
- 簡易パスメジャー ＝ $\frac{３０}{４０}$ ＝ ７５（%）

不透明な管理から定量的管理へ

単体テストの現状　・全分岐のパスの一部しかテストができていない
・全パスの中、どの程度をテストしたか不透明

「自動テスト方式」により大幅なプログラム品質向上が期待できます

定量的管理が可能

第一段階：・全分岐のパスを対象とするテスト方式の導入
・定量的管理：簡易パスメジャー

第二段階：・全パスを対象とするテスト方式の導入
・定量的管理：パスメジャー（品質評価値）

不透明な管理から定量的管理へ：第二段階　全パスの定量的管理

$$\text{パスメジャー} = \frac{\text{実行パス数（N）}}{\text{全パス数（M）} \times \text{有効パス比率の傾向分析値（H）}}$$

・問題：全パス数が天文学的数字になる

（例）・全パス数　$M = 7^{100} \times 14^{3000}$　　（10^{12} ＝1兆）

$$\mathrm{Log}\,10^{M} = 100\,\mathrm{Log}\,10^{7} \times 3000\,\mathrm{Log}\,10^{14} = 290{,}511$$

・実行パス数　N ＝ 17,500

$$\mathrm{Log}\,10^{N} = \mathrm{Log}\,10^{17500} = 4.243$$

・有効パス比率の傾向分析値　H ＝ 0.64

$$\text{パスメジャー} = \frac{4.243}{290511 \times 0.64} = 2.28 \times 10^{-5}$$

不透明な管理から定量的管理へ

単体テストの現状　・全分岐のパスの一部しかテストができていない
・全パスの中、どの程度をテストしたか不透明

「自動テスト方式」により大幅なプログラム品質向上が期待できます

定量的管理が可能

第一段階：・全分岐のパスを対象とするテスト方式の導入
・定量的管理：簡易パスメジャー

第二段階：・全パスを対象とするテスト方式の導入
・定量的管理：パスメジャー

テスト後のプログラムのバグの発生情報を共有することにより、プログラム品質管理の精度が向上していきます

終りに

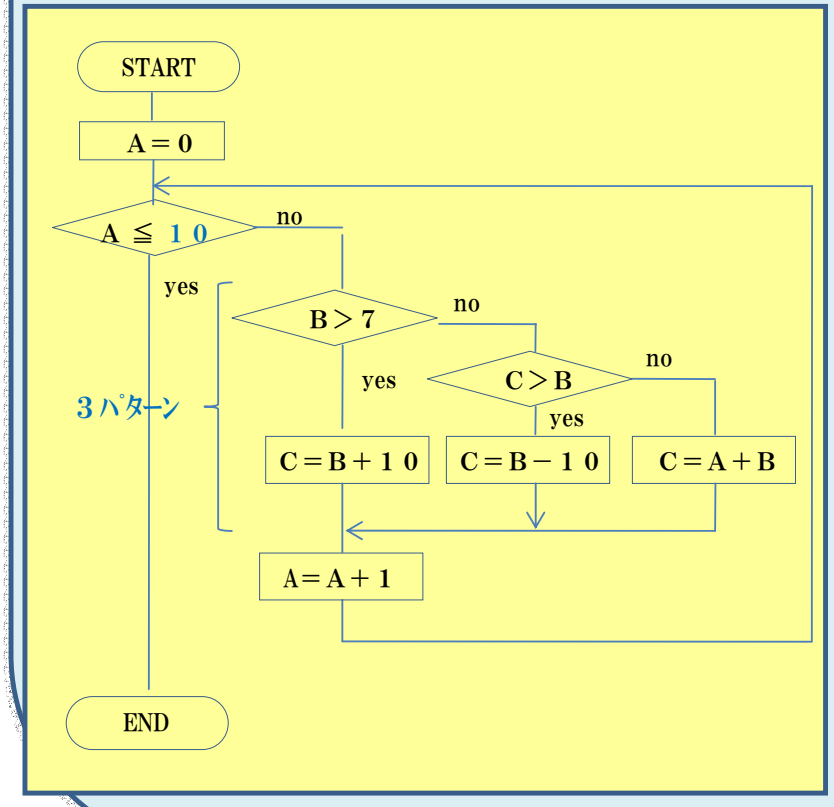
「自動テスト方式」プロトタイプの適用事例

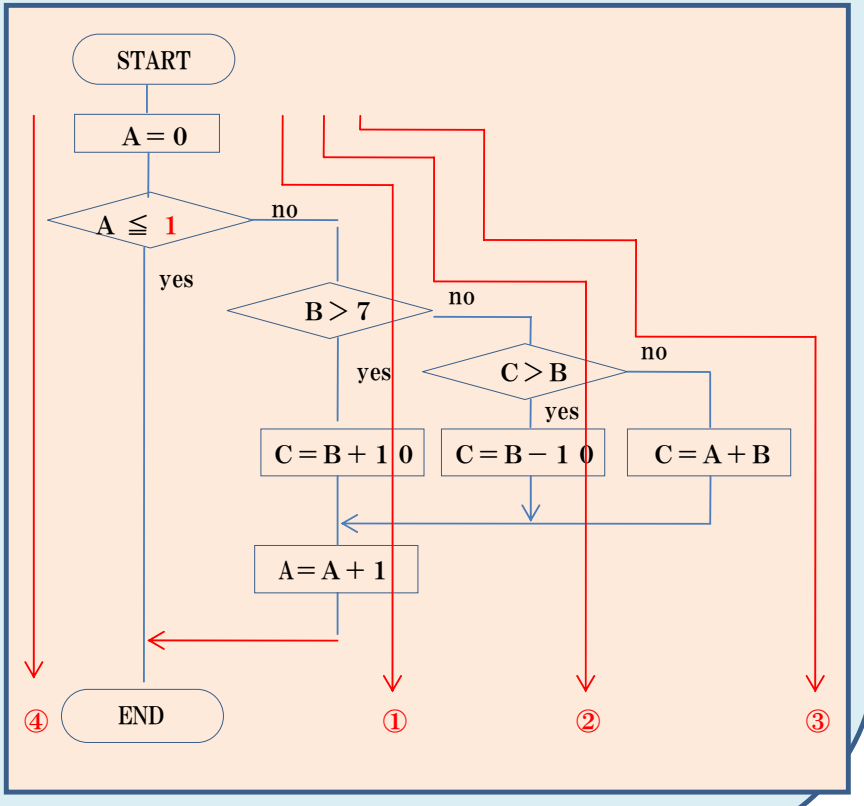
繰返し回数　10回

全パス数：５９，０４９

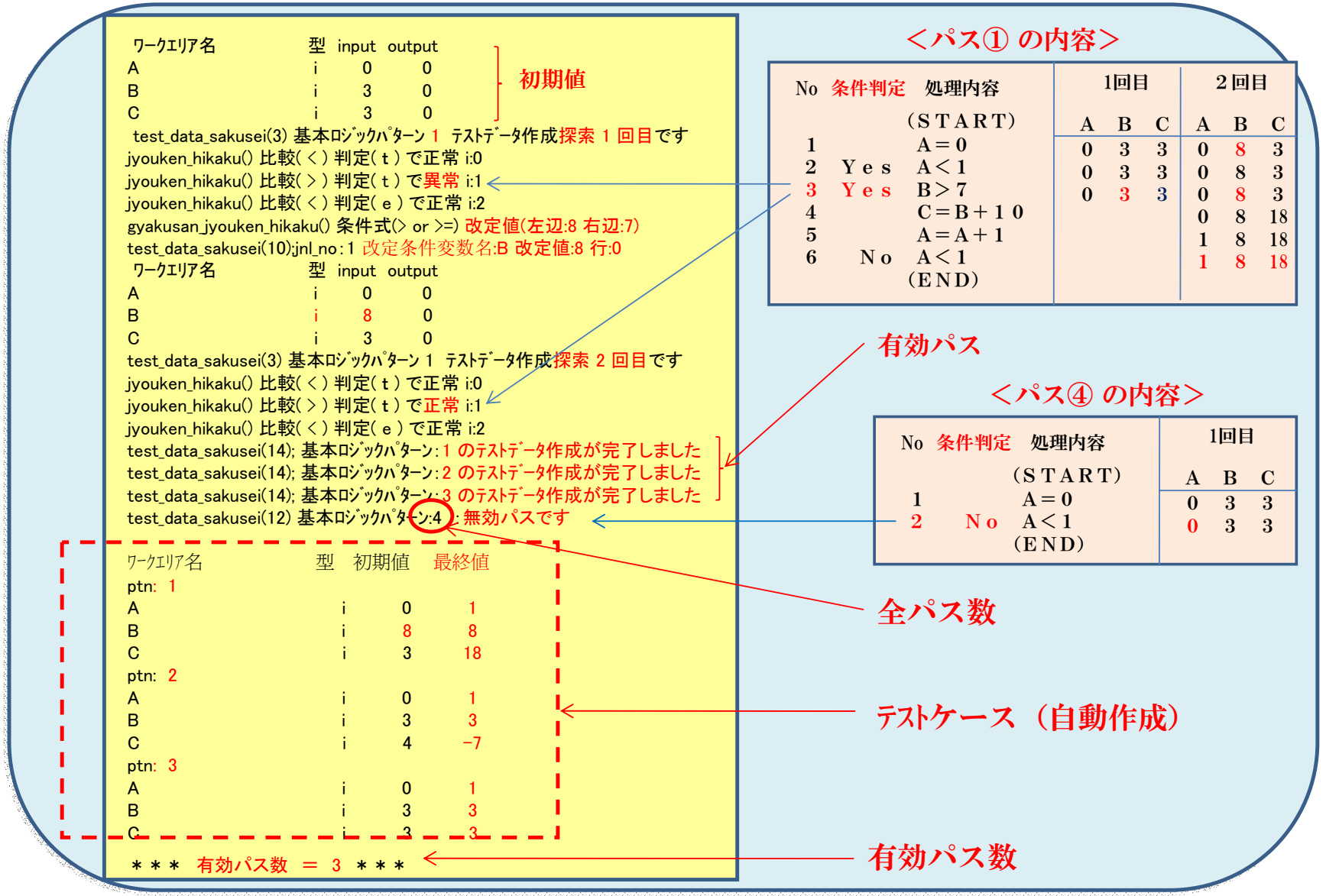
繰返しレベル１（繰返し回数　１回）

全パス数：４





終りに：「自動テスト方式」プロトタイプの適用事例



終りに

品質に対する社会の目は厳しくなるばかりです

「自動テスト方式」によるプログラム品質の向上

システム品質の向上

プログラム品質管理精度の向上

納期管理精度の向上

テストケースの自動作成

プログラム開発作業効率の向上

単体テスト支援ツール「パス解析システム」

－　パスに基づく真の品質管理へ向けて　－

ご静聴ありがとうございました

オーエスエー・リミテッド

代表　新原　俊一

お問合わせ先：ｓ.ｎｉｉｈａｒａ@ｏｓａ－ｌｔｄ．Ｃｏｍ