Concolic TestingツールCREST利用方法の提案
– リグレッションテストへの適用に向けて –

第5分科会

主査　　：奥村 有紀子　（有限会社デバッグ工学研究所）
副主査　：秋山 浩一　　（富士ゼロックス株式会社）
副主査　：堀田 文明　　（有限会社デバッグ工学研究所）
研究員　：村上　仁　　（株式会社ニコン）

目次

0.　Concolic Testingの説明　(1/2)

■ Concolic Testing概要

Symbolic Execution(記号実行)と
Concrete Execution(具体値実行)により
可達パスを通るテストケースを自動生成し、
コンパイルした実プログラムにて実行する、
プログラムを自動解析して、テストを自動実行するテスト技術
→従来のソフトウェア検証を変える大きな可能性
→世界的に利用技術の研究が進められている

0.　Concolic Testingの説明　(2/2)

■ Concolic Testingツール

- C言語：CUTE，CREST，　・・・
- java：jCUTE，CATG，Java PathFinder，・・・
- javascript：Jalangi ，・・・
- .NET：Microsoft Pex,
- LLVM：KLEE

1.　研究目的

■研究目的

・Concolic Testingの利用技術の調査・習得

→まとまった一連の利用解説が見つけられなかった

・現場のリグレッションテストの課題

→テスト不足によるバグの検出漏れがあり，
　効果的なリグレッションテストが実施できていない

Concolic Testingのリグレッションテストへの利用方法を体系的にまとめる

2.　CRESTの機能

■CREST(C言語対応)を研究対象に選択

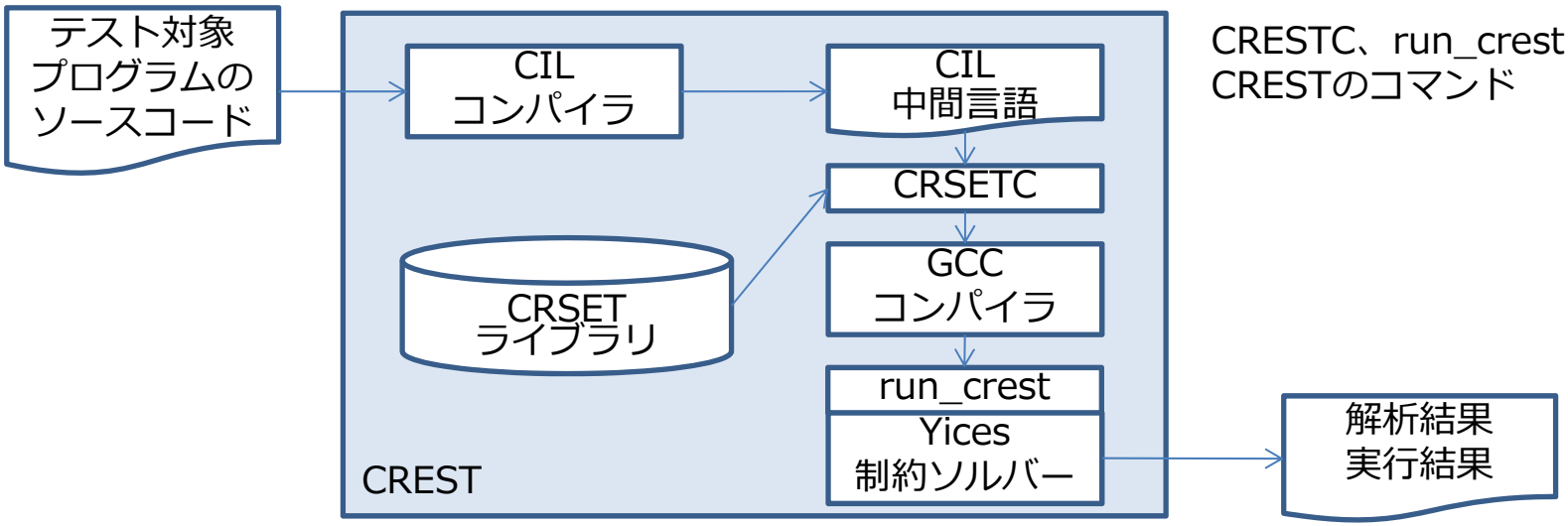
・結果の検証がしやすい

→ソースコード自体を扱うので、別途モデルを作成する必要がない

→生成されたテスト入力と通過した制御パスに関する情報や実行結果についても記録が残る

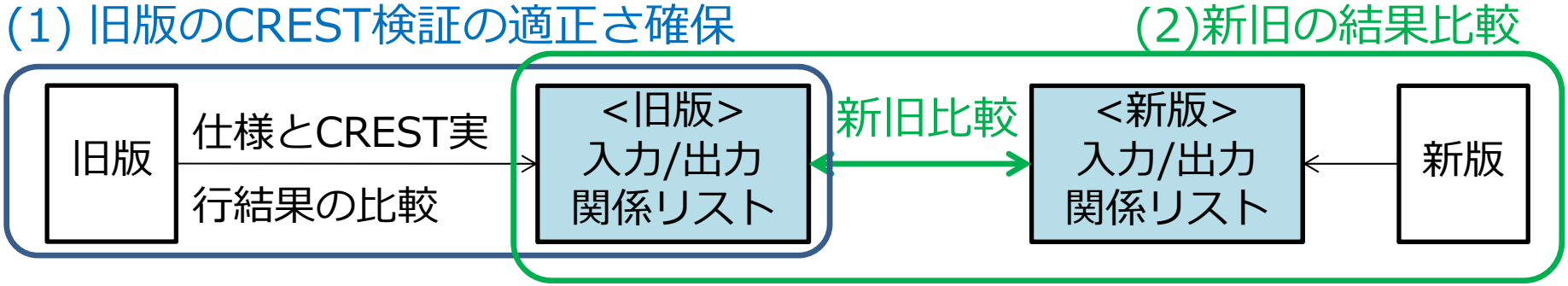
・オープンソースである

■CRESTのイメージ図



3.　リグレッションテストへ利用するための道筋

(1)旧版におけるCRSET検証の適正さ確保
　①パスの網羅性
　②仕様との一致性確保
　③バグの摘出
(2)旧版と新版のテスト結果比較方法の確立

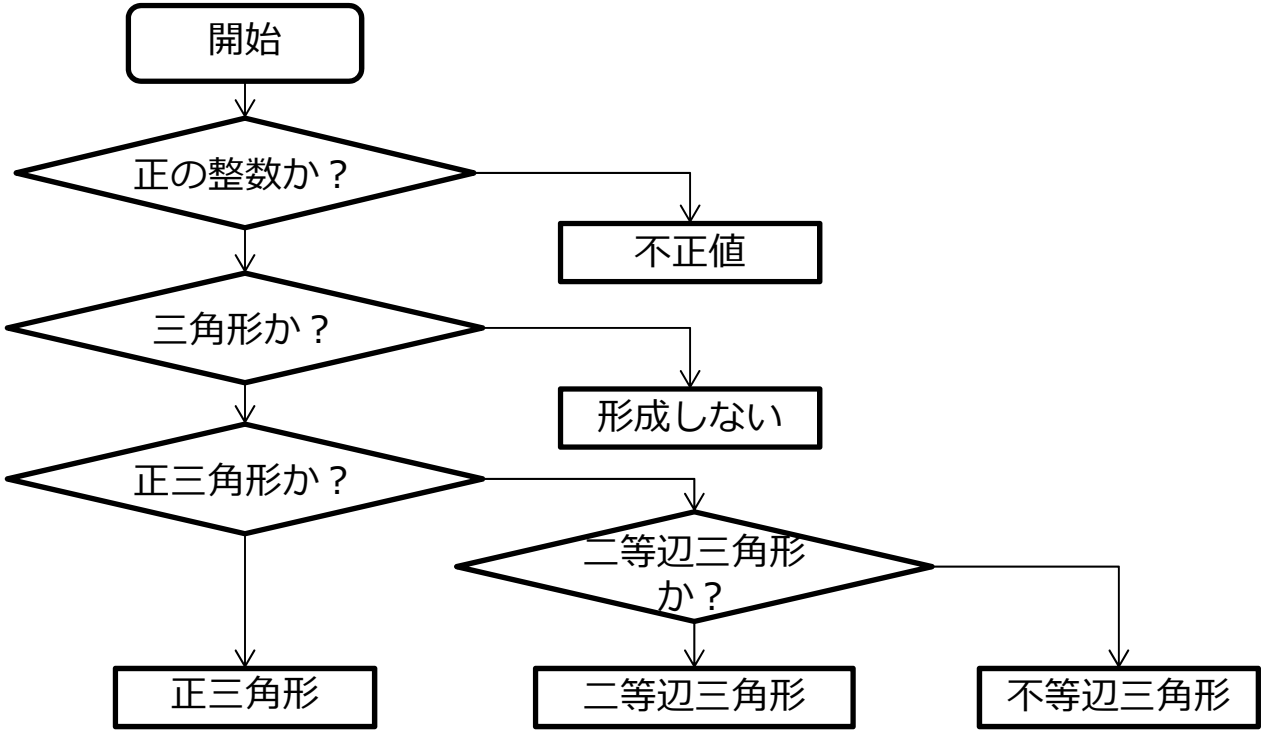


*CRESTで実行可能なテストは仕様・論理の検証である。
　全てのリグレッションテストが代替できるわけではない。

4.　旧版におけるCREST検証の適正さ確保　(1/3)

■三角形形状判定プログラム

実プログラムへの適用には、
多くの分析と準備が必要なため、モデルとして
三角形形状判定プログラムを作成して方法を探った。



4.　旧版におけるCREST検証の適正さ確保　(2/3)

■三角形形状判定プログラム

```
/*  Concolic-Testing三角形の形状判断  */

#include <crest.h>
#include <stdio.h>
#include <assert.h>
#define DEBUGMD

int triangle(int iSide1, int iSide2, int iSide3);
int main(void){
        /* mainは関数を呼び出して結果をプリント */
        int a,b,c,m;
#ifndef  DEBUGMD  /*  CREST実行時はスキップ */
        printf("input 3 sides value =>");
        scanf("%d %d %d",&a,&b,&c);
#endif
        m = triangle(a,b,c);
        printf("Shape of Triangle = %d¥n", m);
        return 0;
}
/*  形状を判定する */
int triangle(int iSide1, int iSide2, int iSide3){
        int msg;
        CREST_int(iSide1);
        CREST_int(iSide2);
        CREST_int(iSide3);
```

CREST実行用に挿入

CREST実行用に
Symbolic変数を宣言挿入

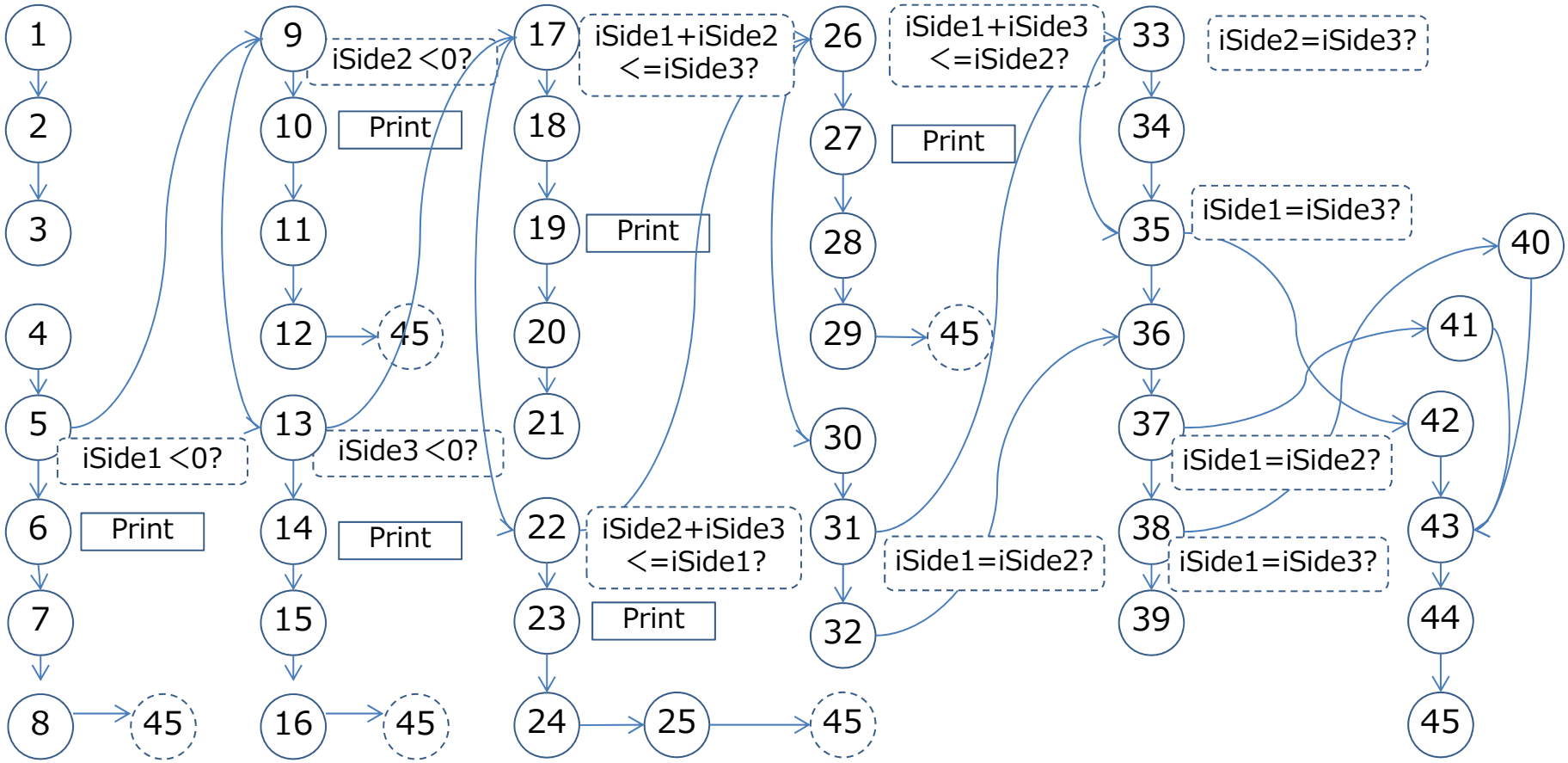
次ページへ

4. 旧版におけるCREST検証の適正さ確保 (3/3)

```
/* 入力データ誤りのチェック */
        if(iSide1 <= 0 || iSide2 <= 0 || iSide3 <= 0)     {
                printf("Input Data is invalid.¥n");
                msg = 10;
                return msg;
        }
 /* 三角形であることのチェック*/
        if ((iSide1 + iSide2)<= iSide3 ||  (iSide2 + iSide3)<= iSide1 || (iSide1 + iSide3)<= iSide2)
        {
                printf("Sides do not form a legal triangle.¥n");
                msg = 10;
                return msg;
        }
 /* 三角形の形状判定*/
        else{
           msg = 20; /* 不等辺三角形  */
                if( iSide1 == iSide2 || iSide2 == iSide3 || iSide1 == iSide3){
                        msg = 30;    /* 二等辺三角形   */
                        if (iSide1 == iSide2 && iSide1 == iSide3){
                                msg =40;    /* 正三角形 */
                        }
                }               return msg;
        }
}
```

4.1　CRSETが実行したパスの解析　(1/2)

■CRESTが解析実行時に出力した中間言語の構造情報



4.1　CRSETが実行したパスの解析　(2/2)

■パスの網羅性確認方法

実行時のブランチカバー情報からブランチとノードの網羅度を把握

◆run_crestコマンドによるテスト実行結果より

Iteration 11 (0s): covered 22 branches [1 reach funs, 22 reach branches].

◆run_crestコマンド実行時の生成された入力値

この値を出力するには、情報収集のための工夫が必要

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	input11
iSide1	0	1	1	1	1	2	1	2	3	4	2
iSide2	0	0	1	1	1	1	2	1	2	3	2
iSide3	0	0	0	1	2	1	1	2	2	2	1

4.2　仕様との一致性確保　(1/2)

■仕様との一致性確保の方法

①仕様をデシジョンテーブルで表現する。

②CREST実行結果から生成された変数と結果をデシジョンテーブルにする。

③両者を比較し妥当性を検証する。

◆仕様とCREST実行結果(テストケース)の比較

仕様の規則	テストケース	対応状況
1	4	対応
2	8，9，11	C=A，B=C，A=Bの3ケースがテストされている.
3	10	対応
4	1	対応
5	2	対応
6	3	対応
7	5，6，7	A＋B≦C，A+C≦B，B+C≦Aの3ケースがテストされている

4.2　仕様との一致性確保　(2/2)

◆仕様のデシジョンテーブル

規則				1	2	3	4	5	6	7
条件部	辺長	A > 0	Y/N	Y	Y	Y	N			Y
		B > 0	Y/N	Y	Y	Y		N		Y
		C > 0	Y/N	Y	Y	Y			N	Y
	三角形形成		Y/N	Y	Y	Y	N	N	N	N
	三辺等しい		Y/N	Y		N				
	二辺等しい		Y/N		Y	N				
動作部	正三角形			X						
	二等辺三角形				X					
	不等辺三角形					X				
	エラー	不当入力					X	X	X	
		三角形非形成								X

◆CREST実行結果のデシジョンテーブル

テストケース			1	2	3	4	5	6	7	8	9	10	11
原因	辺長	A	0	1	1	1	1	2	1	2	3	4	2
		B	0	0	1	1	1	1	2	1	2	3	2
		C	0	0	0	1	2	1	1	2	2	2	1
結果	Invalid		X	X	X								
	三角形非形成						X	X	X				
	正三角形					X							
	二等辺三角形									X	X		X
	不等辺三角形											X	

4.3　バグの摘出検証

■バグの摘出方法

①CRESTを実行すればわかるバグ

→結果の判定により、バグを検知する

②結果は間違いないが途中の処理が適正でない場合

→if文、assert文の挿入により、バグを検知する

■CRESTの入力変数値自動生成の利用

例)ゼロ割の検出

(X+Y)/(Z-2) においてZ-2＝0となるケースがあるか？

→デバッグにCRESTの入力値自動生成を利用する。

→CREST実行時にその文の前にif文またはassert文で

(Z-2)!=0　を挿入する。

5.　旧版と新版の比較　(1/2)

仕様・論理におけるリグレッションテストについて、
旧版と新版のCREST実行結果のDTを比較

	入力値の比較	結果の比較	判定
①	同じ	同じ	OK(デグレードなし)
②	同じ	違う	NG(デグレードあり)
③	違う		変更履歴等の調査必要

*仕様変更がなく旧仕様のままの箇所を比較
*旧版の適正さが確保できている前提
*変更箇所は、旧仕様の場合と同様に検証する

デグレードなし/ありのプログラムについて、
上記でデグレードの有無を判断できることを検証した

5.　旧版と新版の比較　(2/2)

◆新版のCREST実行結果のデシジョンテーブル

新版：辺長の上限判定を仕様追加(辺長≦10)

仕様変更部分

テストケース			1	2	3	4	5	6	7	8	9	10	11	12	13	14
原因	辺長	A	0	1	1	1	11	1	1	1	2	1	2	3	4	2
		B	0	0	1	1	1	11	1	1	1	2	1	2	3	2
		C	0	0	0	1	1	1	11	2	1	1	2	2	2	1
結果	Invalid		X	X	X											
	辺長オーバー						X	X	X							
	三角形非成形									X	X	X				
	正三角形					X										
	二等辺三角形												X	X		X
	不等辺三角形														X	

◆旧版のCREST実行結果のデシジョンテーブル

比較

テストケース			1	2	3	4	5	6	7	8	9	10	11
原因	辺長	A	0	1	1	1	1	2	1	2	3	4	2
		B	0	0	1	1	1	1	2	1	2	3	2
		C	0	0	0	1	2	1	1	2	2	2	1
結果	Invalid		X	X	X								
	三角形非形成						X	X	X				
	正三角形					X							
	二等辺三角形									X	X		X
	不等辺三角形											X	

7. まとめ (1/2)

CRESTで仕様・論理については、以下の方法で
リグレッションテストに適用できることが検証できた

(1)旧版におけるCRSET検証の適正さ確保
- ①パスの網羅性
- ②仕様との一致性確保
- ③バグの摘出

(2)旧版と新版のテスト結果比較

本研究によりConcolic Testingの体系的利用方法を
まとめる目途がついた

課題もあり

6. まとめ－今後の展開 (2/2)

実用には、さらに次の点をクリアする必要があり、
今後さらに研究を続けたい。

(1)CRESTが非対応である型の変換方法調査
(2)大規模プログラムや複数ファイルにまたがるプログラムの扱い方の検証
(3)今回検証していないバグの見つけ方を検証
　例)永久ループ，デッドコード，NULLポインタアクセスなど
(4)Concolic Testingの体系的利用方法をまとめたガイド作成

ご清聴ありがとうございました