Cコンパイラの関数呼び出し規約のランダムテストにおけるエラープログラムの最小化

Minimization of Error Programs in Random Testing of Calling Conventions of C Compilers

中村和博 Kazuhiro Nakamura
石浦菜岐佐 Nagisa Ishiura

関西学院大学 理工学部
School of Science and Technology, Kwansei Gakuin University

1 はじめに

コンパイラのランダムテストは, ランダムに生成したプログラムによるコンパイラのテストを時間の許す限り行うものであり, 様々なテストを経てもなお潜在する不具合を検出する手法の一つである. Quest [1] は関数呼び出し規約を対象としたランダムテストであり, x86 用の GCC-4.0.0 等で不具合を検出しているが, エラーを検出したテストプログラムの最小化を手動で行わなければならないという課題がある. 汎用の最小化ツールとして C-Reduce [2] があるが, 最小化に時間がかかるという課題がある. 本稿では, 関数呼び出し規約を対象としたランダムテストにおけるエラープログラムの自動最小化の手法を提案する.

2 関数呼び出し規約を対象としたランダムテスト

関数呼び出し規約を対象としたランダムテストでは, 図 1 のようなプログラムを生成することにより, 引数と返り値が正しく渡されているかどうかをチェックする. 実際に生成するプログラムは数千行の規模になることがある. このため, エラーを検出した場合, デバッグを行うためには, そのプログラムの最小化 (同じエラーを検出するできるだけ小さなプログラムにすること) が必須となる.

```
01: #include <stdio.h>
02: #define NG(fmt,val) printf("NG("fmt")\n", val)
03:
04: struct S0{ int m0; char m1[2]; };
05: union U0{ struct S0 m0; };
06:
07: int x0 = 0; int f0(char f0_x1);
08:
09: int main( void ){
10:   struct S0 main_x1 = {1,{2,3}};
11:   int f0_ret = f0(main_x1.m1[1]);
12:   if( f0_ret != 4 ) { NG("%d", f0_ret); }
13:   return 0;
14: }
15:
16: int f0(char f0_x2){
17:   union U0 f0_x3 = {{4,{5, 6}}};
18:   if( f0_x2 != 3) { NG("%d", f0_x2); }
19:   return f0_x3.m0.m0;
20: }
```

図 1 最小化の対象とするプログラム

3 エラープログラムの最小化

本稿では, 関数呼び出し規約を対象としたランダムテストにおけるエラープログラムの自動最小化手法を提案する. 本手法における最小化は, 得られたエラープログラムに対し, 以下に示す 1) 〜 4) の操作をそれ以上 1 つでも適用するとエラーが起きなくなるようなプログラムを求めるものである. 操作を適用してもエラーが検出されれば, 更に可能な操作を適用する. 操作の適用によりエラーが検出されなくなれば, その操作を取り消して, 別の操作の適用を試みる.

1) 関数呼び出しの削減: 木構造の関数呼び出しを深さ優先で削除し, 深さ優先で関数のインライン展開を行う.

2) 引数と返り値の削減: 引数と返り値の削除, および余分なエラー判定文の削除を行う.

3) 変数の削減: プログラム中で使われていないローカル変数とグローバル変数をすべて削除する. エラーが消えた場合は元に戻して, 順に一つずつ削除する.

4) 構造体および共用体の簡単化: 使われていない構造体, 共用体を全て削除し, 次に使われていないメンバを全て削除する. どちらの場合もエラーが消えた場合は元に戻して, 順に一つずつ削除する.

4 実験結果

関数呼び出し規約のランダムテストシステムと提案手法に基づく自動最小化を Perl 5.16.3 で実装した. M32R 用の GCC-4.4.1 に対してランダムテストを適用して検出したエラープログラム 5 つに対して最小化を行った. 実験結果を表 1 に示す. “エラーの種類” の ICE は “Internal compiler error” を示す. 提案手法では, 約 2000 行のエラープログラムを 1 分強で最小化できた. C-Reduce に比べると, 最小化を平均で 13.2 倍高速化できた.

表 1 実験結果

プログラム	行数	エラーの種類	時間 [s]	
			C-Reduce	提案手法
error1.c	473	wrong code	353	24
error2.c	110	ICE	29	2
error3.c	606	ICE	148	14
error4.c	1730	ICE	219	32
error5.c	2295	wrong code	1994	76

(Ubuntu 14.04 LTS, Core i7-870, 2.93GHz, RAM 16GB)

5 むすび

本稿では関数呼び出し規約を対象としたランダムテストにおけるエラープログラムの自動最小化を提案した. 今後の課題は, 変数, 構造体, および共用体の削除の際に二分探索を用いるようにすること等があげられる.

参考文献

[1] C. Lindig: “Random Testing of C Calling Conventions,” in Proc. AADEBUG, pp. 3–12 (Sept. 2005).

[2] X. Yang, et al.: “Test-Case Reduction for C Compiler Bugs,” in Proc. PLDI, pp. 335–346 (June 2012).