C コンパイラのテストにおける エラープログラム最小化への式分解の導入

Introducing Expression Splitting in Test Case Minimization for Compier Testing

渡辺 裕貴　　石浦 菜岐佐
Hiroki Watanabe　　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りコンパイラに潜在する不具合の検出を試みる手法である. エラープログラムの最小化は, エラーを検出したプログラムをエラーが起こるできる限り小さなものに縮約する処理であり, デバッグに不可欠である. 従来の手法 [1, 2, 3] は解析木の縮約のみを行うため, 式が長い場合にはその分析が難しくなることがある. これを解決するため, 本稿では最小化に式の分解を導入する.

2　エラープログラムの最小化

エラープログラムの最小化は, プログラムの縮約変換をエラーが生じる限り繰り返し適用することにより行う. 縮約変換の結果エラーが消えたり未定義動作が生じた場合は, その変換を取り消して別の縮約変換を試み, どの変換も適用できなくなったものを最小化の結果とする. 従来手法 [1, 2, 3] は, いずれも解析木のボトムアップな縮約に基づいている. このため, 図 1 の上のように, 最小化後のプログラムが長い式を含む場合, 式の構造がわかりにくく, 分析が難しくなる.

```
static volatile signed int x1 = 0;
volatile signed char x2 = 0;

int main (void)
{
 static unsigned int t1 = 358U;

 t1 = (unsigned int)(((signed long long)(((unsigned char)
      ((signed long)-1L+((signed char)(((signed long long)
      ((signed long long)-9223372036854775808LL-x1))/
      (((signed long long)((unsigned long long)
      11529215046068469 76LLU>>((unsigned short)x2)))/
      (signed char)-1)))))<<(signed long long)1LL))^
      (signed long long)360LL);

 if (t1 != 358U) NG("t1", "%u", t1);

 return 0;
}
```

```
 t1d = 11529215046068469 76>>x2;
 t1c = (-9223372036854775808-x1)/(((signed long long)t1d)/-1);
 t1b = -1+t1c;
 t1a = t1b<<1;
 t1 = t1a^360;
```

図 1　従来手法で最小化したプログラム (上), 分解後の式 (下)

3　式分解の導入

本稿では, 最小化に式の分解を導入することにより, 長い式を含むエラープログラムを分析し易くする. 図 1 の上の長い式を下に示すように 1 式あたりの演算ができる限り少なくなるように分解する. 同時に, 式の値に影響を及ぼさないキャストも削除する.

式の分解は, 式の解析木を根から始めて再帰的に分解することにより行う. 例えば図 2 において, 変数 a に加算結果が代入されていれば, 新たな変数 b を設けて部分式の結果を代入することにより 2 つの式に分解する. この分解の結果エラーが消えればこの分解は取り消す. この操作を可能な部分木全てに対して再帰的に適用することにより, エラーを起こすプログラムで 1 式中の演算子が極小になるものを求めることができる. キャスト演算は, 部分式の型が変化せず, エラーが消えない場合にのみ削除する.

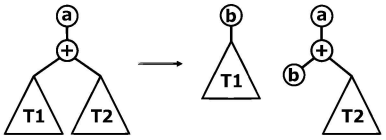

図 2　式の分解操作

4　実装結果

提案手法を Orange4 [3] に Perl5 で実装した. GCC-4.8 のエラープログラム 10 本 (2,000 行~2,500 行) に対し, 最小化に要した平均実行時間を表 1 に示す. “式分解なし” は従来手法, “式分解あり” は本手法であり, 式分解の導入による計算時間の増加は約 8.5% である.

表 1　エラープログラム最小化の平均実行時間

	式分解なし	式分解あり
実行時間 [秒]	11.24	12.20

5　むすび

式の分解に基づくエラープログラム最小化の強化手法を提案した. 現時点では, 定数とスカラ変数を含む式しか分解できないため, 配列, 構造体・共用体, 関数呼び出しを含む式まで扱えるように実装を拡張することが今後の課題である.

参考文献

[1] J. Regehr, Y. Chen, P. Cuoq, E. Eide, C. Ellison and X. Yang: “Test-case reduction for C compiler bugs,” in *Proc. PLDI 2012*, pp. 335–346 (June 2012).

[2] C. Sun, Y. Li, Q. Zhang, T. Gu, and Z. Su: “Perses: Syntax-guided program reduction,” in *Proc. ICSE’18*, pp. 361–371 (May 2018).

[3] S. Takakura, M. Iwatsuji, and N. Ishiura: ”Extending equivalence transformation based program generator for random testing of C compilers,” in *Proc. A-TEST 2018*, pp. 9–15 (Nov 2018).