ハックの自動発見について

土屋 達弘[1,a)]

概要：本稿では，形式手法技術を利用したハックの自動発見について議論する．ここでハックとは，機械語に近いレベルでの高速化を主たる目的としたプログラミング技法のことである．このようなハックをパラメータとして表現できることを示し，有界モデル検査に類似した手法で，ハックが存在するかどうかを問う問題を，SAT や QBF などの問題として扱うという提案手法の概要について記す．

On Automatic Discovery of Hacks

1. はじめに

本稿では，形式手法を応用したハックの自動発見について議論する．ここでハックとは，典型的には文献 [1] で説明されているような，プログラムの実行の高速化を主たる目的とする機械語に近いレベルでの特殊なプログラミング技法を指すものとする．例として，C 言語のプログラムを対象に，符号付き 32 ビット整数変数の絶対値を求める計算を考える．絶対値の計算は，単純なコーディングで以下のように実現できる

```
x > 0 ? x : -x
```

しかし，この方法は条件分岐が必要という問題がある．条件分岐は，パイプライン処理における制御ハザードとなるため，高速化の点では望ましくない．これに対し，文献 [1] では以下の条件分岐を用いない方法を紹介している．

```
y = x >> 31;
z = (x ^ y) - y;
// z: the absolute value of x
```

上記の文献では，このようなハックが多数取り上げられている．一方で，プログラム開発者が必要とする計算を高速に実現したいと考えた場合，高速でかつ既知のプログラミング技法が知られているとは，必ずしも限らない．そこで本稿では，形式手法に関連する技術を用いて，与えられた計算に対するハックを自動的に発見する方法について議論する．

[1] 大阪大学
1-5 Yamadaoka, Suita, Osaka 565–0871, Japan
[a)] t-tutiya@ist.osaka-u.ac.jp

2. 自動発見の方法

提案する手法は，有界モデル検査のアイデアと，SAT, SMT (Satisfiability Modulo Theories), もしくは, QBF (Quantified Boolean Formula) ソルバを用いて目的を実現する．まず，定数個の基本的な演算からなる計算は，パラメータ化して表現できることを例を用いて示す．図 1 は，絶対値を求める前述の 2 通りの方法を実装したプログラムを示している．条件分岐を利用した自明な方法は関数 `g()` で実装されている．

一方，右シフトや排他的論理和を用いた方法は，パラメータ化された形で，関数 `f()` として実装されている．この関数は静的単一代入の形式に従っており，変数は一度しか更新（定義）されない．配列 `a` はこれらの更新される変数であり，`a[0]`, `a[1]`,... の順に更新されている．ただし，`a[0]` と `a[1]` は，0 と引数 `x` の値を表すのに利用しており，実際の演算は 3 回の関数 `s()` の呼び出しにより行われ，結果は `a[2]`, `a[3]`, `a[4]` の順番で保存される．

関数 `s()` はいくつかのビット演算を表現しており，5 個の引数によって，演算の種類と，オペランドとしてそれまでに得られている演算結果，もしくは，定数を用いることを指定することができる．それまでに得られている演算結果とは，`a[`i`]` を更新する場合は，`a[0]`, ..., `a[`$i-1$`]` であるので，0 から $i-1$ までの添え字を指定する．その結果，条件分岐を用いない絶対値の計算が計 15 個のパラメータによって表現されている．

以上から，3 回の基本的な演算で絶対値を求めるプ

```
#include <assert.h>
int a[5];

enum instruction
  {or, xor, and, shiftr, shiftl, add, sub};

int g(int x) {
  return x > 0 ? x : -x;
}

int f(int x) {
  a[0] = 0;  // 0
  a[1] = x;  // input
  a[2] = s(shiftr, 1, 0, 0, 31);
  a[3] = s(xor, 1, 0, 2, 0);
  a[4] = s(sub, 3, 0, 2, 0);
  return a[4];
}

int s(enum instruction ins, int v1, int add1,
int v2, int add2) {
  int rv;
  switch (ins) {
  case or: rv = (a[v1] + add1) | (a[v2] + add2); break;
  case xor: rv = (a[v1] + add1) ^ (a[v2] + add2); break;
  case and: rv = (a[v1] + add1) & (a[v2] + add2); break;
  case shiftr:
    rv = (a[v1] + add1) >> (a[v2] + add2); break;
  case shiftl:
    rv = (a[v1] + add1) << (a[v2] + add2); break;
  case add: rv = (a[v1] + add1) + (a[v2] + add2); break;
  case sub: rv = (a[v1] + add1) - (a[v2] + add2); break;
  default: rv = 0;
  }
  return rv;
}

int main() {
  int a = -10;
  assert(f(a) == g(a)); // must hold for any a
  return 0;
}
```

図 1 絶対値を計算する二つの関数（g(): ゴールデンモデル．f(): パラメータで表現されたハック．）

ログラミング技法を発見する問題は，$f(a) = g(a)$[*1] が $-2^{31} \leq a \leq 2^{31}-1$ の範囲で成り立つような，15 個のパラメータを求める問題として表現できる．a[i]($i \in \{2,3,4\}$) に対する s() の呼び出しのパラメータを $P_{i,1}, ..., P_{i,5}$ と表し，計 15 個のパラメータ $P_{2,1}, ..., P_{4,5}$ によって実現された関数 $f()$ を $f_{P_{2,1},P_{2,2},\cdots,P_{2,5}}()$ と表記すると，この問題は以下の式が真となるような $P_{2,1}, ..., P_{4,5}$ を求める問題といえる．

$$\exists P_{2,1}, ..., P_{4,5},$$
$$\forall a, -2^{31} \leq a \leq 2^{31}-1 : f_{P_{2,1},...,P_{4,5}}(a) = g(a)$$

ただし，$i \in \{2,3,4\}$ について

$$0 \leq P_{i,2} \leq i-1,\ 0 \leq P_{i,4} \leq i-1,$$
$$P_{i,2} \neq 0 \Rightarrow P_{i,3} = 0,\ P_{i,4} \neq 0 \Rightarrow P_{i,5} = 0$$

という制約を設ける．この式を SAT ソルバ等のツールで解ける形で表現することで，ハックの自動発見という目的が実現できると予想している．

3. 実現にあたっての課題

まず，プログラムで表現されている関数 $f(), g()$ を，ブール式などの数式の形で表現する必要がある．この処理には，有界モデル検査で用いられているプログラムからブール式への変換方法を利用することができる [2].

反対に，有界モデル検査にはない課題が存在する．有界モデル検査ではモデル検査を SAT （充足可能性判定問題）を解くことで実現するが，SAT は変数を束縛する限量子がすべて存在記号 (∃) であるように制限された QBF と見なせる．一方，ここでの問題は，上式の様に，限量子として存在記号と全称記号 (∀) とを含む式が真になるかどうかを問うことである．したがって，SAT ソルバをそのまま利用することはできない．

この課題への解決策の一つは，SAT ソルバではなく，QBF ソルバ，あるいは，限量子を扱うことができる SMT ソルバを用いることである．

もう一つの方法は，可能な入力を一つ一つ固定して式をつくり，異なる入力それぞれについてその式を連言節とする全称記号のない大きな式を生成し，SAT ソルバを利用できるようにする方法である．ただし，この方法だと 1 ワードのビット数がかなり少なくない限り，可能な入力の総数が膨大になり，一つの式として表現することが困難なことが予想される．そこで，解決策として，1 ワードを 8 ビット程度として SAT ソルバを用いて自動的に得られたハックを，1 ワード=32 ビットなどの現実的な場合に適合するように改変することが考えられる．

4. おわりに

本稿では，形式手法技術を応用したハックの自動発見について，提案する方法の概要について簡単に述べた．今後は，実際に実現可能か，また，実現できた場合はどの程度複雑なプログラミング技法が発見できるのか，有用な未知のハックを発見できるか，といった点で研究をすすめたい．

参考文献

[1] Warren H. G.: *Hacker's Delight, Second Edition*, Addison-Wesley Professional (2012).
[2] Clarke, E., Kroening, D., and Yorav, K.: Behavioral consistency of C and verilog programs using bounded model checking, In Proceedings of the 40th annual Design Automation Conference (DAC '03), 368-371 (2003).

[*1] プログラムでの関数 f(), g() が実装している数学的な関数を $f(), g()$ と表す．