C言語における部分評価手法の提案とその実現

1J－8

岩本 奈美†　山本 晋一郎‡　阿草 清滋§

†名古屋大学大学院工学研究科　‡愛知県立大学情報科学部　§名古屋大学情報メディア教育センター

1 はじめに

部分評価は依存解析によりプログラムの入力値に関して評価できる部分とできない部分に分け、評価できる部分に関しては実行することによって、評価できない部分のみが残った効率的なプログラムを作成することである。

本研究では、C言語の関数を対象として部分評価を行なうことを考える。C言語を対象とした部分評価は[2]で研究されている。この部分評価手法を改良し部分評価器を実装した例と問題点などを示す。

2 部分評価とは

2.1 Binding Time Analysis

部分評価を行なう単位は関数とする。部分評価をする対象は関数の引数と関数内のローカル変数とする。関数内の変数について静的に値が決まるもの(static)と、動的に値が決まるもの(dynamic)と分類を行なう。この解析をBinding Time Analysisと呼ぶ。静的か動的かの解析は、引数の入力値とローカル変数の初期値を入力値として、評価できる変数と評価できない変数に分類を行なっていく。

2.2 Generating extensions

Binding Time Analysisによって解析を行なったプログラムを入力として、generating extensionプログラムを出力するためにgenerating extension生成器を実行する。この生成器は、staticな部分を評価するためにgenerating extensionプログラムに出力し、dynamicな部分を最終プログラムである特殊化プログラムに残すようにコード生成関数に変換してgenerating extensionプログラムに出力する。この出力されたgenerating extensionプログラムにstaticな引数の値を入力してコンパイラで実行することによりstaticな部分は評価されdynamicな部分のみが残った特殊化プログラムが生成される。

Partial Evaluation for the C Language and its Application
Nami Iwamoto†　Shinichiro Yamamoto‡　Kiyoshi Agusa†
†Nagoya University　‡Aichi Prefectural University

2.3 Binding Time Analysisの問題点

Binding Time Analysisでは、変数に対してstaticかdynamicかということを決めている。この場合staticな変数にdynamicな変数が代入された場合、その変数はdynamicであるとみなされてしまう。この場合、特殊化プログラムに評価できる部分が残ってしまうことになり、正確に部分評価ができなくなってしまう。1つの変数に対してstaticかdynamicかのどちらかに分類してしまうのではなく、staticでもありdynamicでもあるような変数という分類も必要である。

3 Binding Time Analysisの改良

dynamicなwhile文の中の文は、依存解析によってstaticと分類されても何回ループするか分からないのでdynamicとして扱う必要がある。このような場合、この変数をdynamicとするのではなく、staticでもdynamicでもある変数として扱い、dynamicになる直前まではstaticとして評価を行ない、dynamicな文になる直前で変数の値を代入する文を追加するということを行ない評価できる部分を正確に分類することが可能となった。

4 部分評価器 SPE

前節の手法を用いて実装した部分評価器SPE (Sapid Partial Evaluation)の構成を図1に示す。

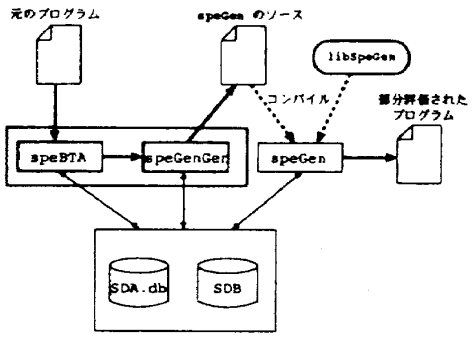

図1: 部分評価器SPE

4.1 speBTA

SPEBTA は、部分評価を行なう関数名が与えられ、実際に部分評価を行なう変数はプログラムの関数呼び出し式の引数の値から決める。この引数の値を元にして関数内の Binding Time Analysis を行なうことによって関数内の式について static か dynamic に分類する。このとき、static か dynamic どちらかに分類できない変数については途中で dynamic な変数に変化するので、static と dynamic との両方に入れておく。

4.2 speGenGen

speBTA の情報を利用して元のプログラムから生成プログラム (speGen) のソースプログラムを作成する。この部分を speGenGen と呼ぶ。この生成ソースプログラムは static な文はそのまま出力され、dynamic な文は最終結果の評価後ソースプログラムに出力されるような関数呼び出し式が出力される。また、for や while などのループに関しては if、goto の文に書き換えることにより評価する部分を多くしている。

4.3 libSpeGen

生成ソースプログラムをコンパイルし libSpeGen とリンクすることによって生成プログラム (speGen) が作成され、これを実行すると部分評価された最終結果となるソースプログラムが生成される。ライブラリ libSpeGen は、dynamic な文を評価後ソースプログラムに出力するための関数群が用意されている。ここで、関数内で呼び出している関数の引数が static な変数である場合、最終の特殊化プログラムでは引数に定数が入っている必要がある。このために dynamic な文を出力する時にその中に static な変数がある時は変数を現在の値に変換して出力している。

5 実行例

図 2のプログラムの関数呼び出し pow(x,5) に関して部分評価を行なった実行例を示す。下線が static な変数である。

図 2のプログラムに対して speGenGen を実行し生成された生成プログラムのソースプログラムを図 3に示す。ここでは static な変数がソースにそのまま出力され、dynamic な変数が最終プログラムに残るように関数呼び出しに変換されている。speGen を実行することによって生成されたソースプログラムを図 4に示す。これにより dynamic な文だけが残ったプログラムが生成される。

```
int pow(int base, int n)
{
  int pow;
  pow = 1;
  while(n--)
    pow *= base;
  return pow;
}
```

図 2: speBTA で解析したプログラム

```
pow_gen()
{
  int spe_label;
  int n = 5;
  speRegisterStaticVar(4194305, &n);
  speFuncHeader(4194306, 1, 3145728);
  speDecl(2097154);
  spePendInsert(6291460);
  while ((spe_label = spePending())
      != SAPID_NON_ID) {
    switch(spe_label){
    case 6291460:
      speState(7340034);
      spePendInsert(6291458);
      break;
    case 6291458:
      if (n--)
        spePendInsert(6291457)
      else
        spePendInsert(7340041)
    case 6291457:
      speState(7340039);
      spePendInsert(6291458);
      break;
    case 7340041:
      speState(7340041);
      spePendInsert(-1);
      break;
    }
  }
}
```

図 3: 生成プログラムのソースプログラム

```
pow_spe(int base)
{
  pow = 1;
  pow *= base;
  pow *= base;
  pow *= base;
  pow *= base;
  pow *= base;
  return pow;
}
```

図 4: 部分評価されたソースプログラム

6 問題点と今後の課題

現在、static な変数の解析を行なっているのはスカラー変数のみである。よって、配列の要素や構造体のメンバやポインタといったものについてはすべて dynamic として扱っている。このような複雑なデータ構造を持つ変数に対しての解析を進める必要がある。また、部分評価を行なう場合に、どの関数について部分評価を行なうとよいのかをユーザに提示できるようなツールを用意する必要がある。同じ関数内でも部分評価を行なう意味のある引数を調べるということも必要である。

参考文献

[1] Jones, Gomard and Sestoff, "Partial Evaluation and Automatic Program Generation", Prentice Hall, 1993

[2] Lars Ole Andersen, "Program Analysis and Specialization for the C Programming Language", Ph.D. Thesis, May 1994