ISS-A-021

スタック計算機の末尾再起最適化とコンパイル可能性について

氏家　雄司†　　塩谷　勇††
† 法政大学理工学部創生科学科　　†† 法政大学理工学研究科

1. はじめに

本報告では、末尾再起最適化(Tail Call Optimization, TCO)の命令 TCO を持つスタック計算機の末尾再起最適化について述べ、C 言語風の手続き型言語のコンパイルの可能性について報告する。

一般に、再起は繰り返しよりもプログラムを考える上で利用者にとって有効である。この研究では、利用者は再起呼び出しによるメモリのオーバーフローを意識することなく、プログラムを記述できる範囲について検討してきた。アルゴリズムを考える上では再起は有効であるが、それを繰り返しのアルゴリズムへの変換は難しいが、末尾最適化が可能ならば、利用者にとって考えることと実装の距離が極めて近くなる。TCO の実装については、JavaScript 言語でも時間をかけて検討されている[2]。

本報告では関数の値呼び出し(call by value)のみを考える。末尾最適化は、return 文からの関数呼び出しの際、新たにスタックフレームをプッシュダウンすることなく、現在のスタックフレームを上書きして再使用する。従って、古いスタックフレームは削除されるから、現在利用している変数の記憶域が削除されるため、つぎの対策が考えられる。

(1) 仮引数の記憶域を利用した手法、例1。
(2) 引数を追加するプログラム変換の手法、例 2。
(3) CPS (Continuation Passing Style) の手法
(4) gcc の実装のように関数を展開する手法
(5) 動的にメモリを確保して、不要になれば GC で回収

本報告では、TCO 命令を持つスタック計算機の(1)仮引数の記憶域を利用した手法、(2)の引数を追加する手法について検討を行い、それらのコンパイル技法について述べる。

2. スタック計算機 STM

スタック計算機[1]は計算機の構造とコンパイラの学習のために利用されているシステムで、その特徴は TCO 命令を持つことである。スタック計算機の TCO 命令の実装は極めて簡単で、関数を呼ぶ命令 CALL と比較してみよう。

CALL n :次の命令の番地をスタック S[sp]に、sp←sp+1, S[sp]←bp, sp←sp+1, bp←sp, n 番地の P-CODE に jump

TCO n : sp←bp, n 番地の P-CODE に jump

sp はスタックポインタ、bp はベースポインタである。n は P-CODE の番地で、単に n 番地に jump するのみである。sp←bp から、スタックフレームが上書きされることがわかる。関数の戻り番地や前のスタックフレームへのリンクもそのままでよい。

「例 1」TCO を利用した繰り返しの呼び出し例。仮引数の記憶域に引き続き使用する値を保持することで、スタックフレームを削除しても正しく計算が行われる。

```
x=f(3);
function f(x) {
 if x==0 return 1;
 return f(x-1)
}
```

3. 仮引数の記憶域を利用した手法

「例 2」1+2+3+4+..+10 を再帰的に計算する例。

```
x=f(10);
function f(x) {
 if(x==0) return 0;
 return x+f(x-1)
}
```

上記の計算では、関数の仮引数を増やすことで、関数値を増やした引数に加えることで逐次和を求めることができる。しかし、関数の引数が変わるために、(3)-(5)と同様の欠点である分割コンパイルができなくなる。

```
function f(x,y) {
 if(x==0) return y;
 return f(x-1,y+x-1)
};
x=f(10,10)
```

4. コンパイラの実装

スケルトンに基づいた手法を実装は、TCO が可能なパタンを用意し、条件を満たせば最適化を行うものである。本手法は、式、代入文、四則演算に限定し、(a)変数が仮引数か、または、局所変数、(b)代入文の左辺、または、右辺、(c)return 文の式のパタンに分類してスケルトンを構成する。

参考文献

[1] Isamu Shioya, Stack Machine with Tail Call Optimization, http://shio2.k.hosei.ac.jp/
[2] ECMAScript, https://www.ecma-international.org/.