ISS-A-004

スタック・マシンによる末尾再帰最適化の考察

Stack Machine and Tail Recursive Optimization

吉村　圭右　　塩谷　勇

Keisuke Yoshimura　Isamu Shioya

†法政大学理工学部

Faculty of Science and Technology, Hosei University

1　はじめに

・スタック計算の作成
・末尾再帰最適化の実装
・CPSの実装の提案

理想は、プログラム作成者は再帰によりプログラムを表現する一方で、コンパイラは再起を回避した繰り返しのプログラムに変換することである。これは、スタック・フレームの再使用による末尾再帰最適化(Tail Recursive Optimization)により実現できる。しかし、すべての再起プログラムについて、末尾再帰最適化技法が適用できるわけでない。

2　研究目的

再帰プログラムと同じ計算をする繰り返しのプログラムに変換する。再帰プログラムは計算時にスタックが成長する一方で、繰り返しはスタックの成長がしない。

しかし、すべてのプログラムに適用することが可能であるというわけではない。プログラムを書き換えて、stack frameを成長させない条件や範囲などの考察を行う。

これらを踏まえて、本研究では、スタック・マシンに末尾再帰最適化の命令TROを実装することで、末尾再起最適化の適用範囲について考察し、またCPSの実装の提案することが本研究の目的である。

3　研究方法

関数の計算時にstack frameを成長させない命令を実装することで研究を行う。

(a)関数引数の値を上書きして、stack frameを再使用。
(b)プログラムの書き換え。
(c)lambda式の導入。

4　研究結果

上記に記載した(a)、(b)、(c)の方法を用いて研究を行った。その結果を以下に記載する。

```
function g(x,y){
 return x+y
}
function f(x,y){
 z=x +y+1;
 return g(z,y)+x
};
x=f(2,3)
```

⇨

```
function g(x,y,z){
 return x+y+z
}
function f(x,y){
 z=x +y+1;
 return g(z,y,x)
};
x=f(2,3)
```

通常は無名関数
lambda式を導入

図1　関数を導入した場合

左の命令文では、return文の中には関数引数のほかに変数が存在するため、TRO命令をするとxがスタックフレームを使用する際にg(z,y)が使用したスタックフレームが上書きされ、g(z,y)の値が消滅してしまう。これは右の命令文のようにlambda式の導入することで回避することができ、適用範囲内となった。

```
function f(x,y) {   ←仮引数が2つ
z=x+y; z=z*z;
return z
};
function g(x,y) {  ←仮引数が2つ
return f(x+1,y)*f(x,y+1);
};
x=f(g(2*4,5+2)+1,g(5,5))
```

図2　関数を2回呼び出す場合

この場合、関数を2回呼び出すことで、f(x,y+1)計算する際にf(x+1,y)が計算に使用したスタックフレームを使用してしまうため、f(x+1,y)の値が上書きされ消滅してしまいできなかった。これは(a),(b),(c)の方法ではできないが、TRO命令の拡張を行うことで可能となることがわかった。

5　まとめ

図1は、return文の中には関数引数のほかに変数が存在するため、g(z,y)が使用したスタックフレームが使用され、g(z,y)の値が消滅してしまう。これはlambda式の導入することで回避することができ、適用範囲内となった。

図2は、関数を2回呼び出すことで、f(x,y+1)計算する際にf(x+1,y)が計算に使用したスタックフレームを使用してしまうため、f(x+1,y)の値が上書きされ消滅してしまいできなかった。これは(a),(b),(c)の方法ではできないが、TRO命令の拡張を行うことで、f(x+1,y)の値が上書きされることを回避できる可能性がある。

参考文献

[1] ECMAScript6, webkit.org. 2015.