7D−3

Datarolマシンへの高階関数及び遅延評価の実装方式

河内 久和　高橋 英一　谷口 倫一郎　雨宮 真人
九州大学総合理工学研究科

1　はじめに

関数型言語は、記述の簡潔さ、並列性抽出の容易さなどの優れた面を持っている。関数型プログラミングにおいて関数を引数にしたり、関数を結果として返す高階関数を用いるとプログラムの記述が非常に簡潔になり、全体構造が理解しやすくなるという利点がある。また、遅延評価を導入するとデータ駆動型方式では難しいとされる無限リスト操作などを行なえるという利点がある。本稿では、筆者らが関数型言語の処理系を実現するアーキテクチャーとして開発しているDatarolマシン[1]への高階関数及び遅延評価の実装方式について考察する。

2　高階関数の実装方式

2.1　記述方式

Datarolマシンの高級言語として関数型言語Valid[2]を用いている。しかし、現仕様では、インタープリタで実行させるために高階関数の記述にリスト表記を用いており、複雑な表記になっている。高階関数の記述を簡潔にするために新しくラムダ表記を導入することにした。例えば、ある関数を2回適用するTwiceという高階関数は次のようにValidで表記する。

```
function Twice(f:function) return (function)
= lambda(x)f(f(x)) ;
```

2.2　実装方式

高階関数の実装方式として2つの方式を提案する。

2.2.1　SM recipe 方式

Validで書かれたプログラムからDatarolグラフを生成するDatarolコンパイラは、Twiceのようなラムダ表記の場合は、まずラムダ・リフティングを行ない以下のようにする。

Twice (f) = β (f)
β (f,x) = f(f(x))

Twice(f)は、β(f,x)の部分適用とみなされ、コンパイラは、recipeを構築するコードを出す。recipeとは、図1に示すように、完全適用の関数コードへのポインタと引数とのペアであり、SM(構造体メモリ)のアレイ部に作られる。部分適用の実行は、このrecipeへのポインタを返す。

また、完全適用(β(f,x))のDatarolグラフは、次に示すzcall,zlink,zrlinkという3つの新しい命令を導入した図2になる。

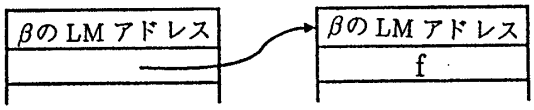

図1: Twice(Twice(f)) のrecipe

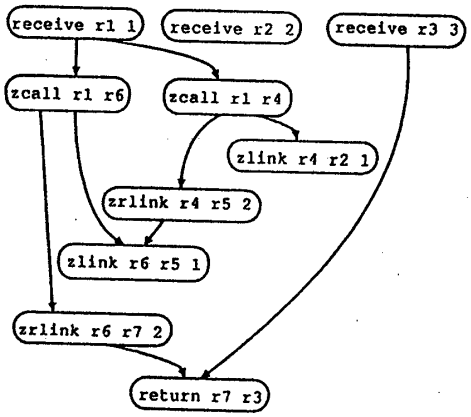

図2: (β(f,x)) のDatarolグラフ

zcall ：オペランドのレジスタのr-tag（レジスタがrecipeへのポインタなら1、そうでなければ0）の値によって以下の動作をする。

A　r-tagが0の場合
関数インスタンスにレジスタファイル（作業領域）を確保する

B　r-tagが1の場合
関数インスタンスにレジスタファイルを確保し、recipeの指す関数コードのreceive命令にrecipe中の引数を渡す

zlink ：関数インスタンスに

A　関数の引数番号目の引数を渡す

B　関数の引数番号+α目の引数を渡す。
（αはrecipe中の引数の数）

zrlink ：関数の結果値の送り返し先情報を

A　引数番号目の引数として送る

B　引数番号+α目の引数として送る

2.2.2　インスタンス・コピー方式

2.2.1の方式では、パラメータをrecipeに蓄えていき、パラメータがすべて揃った時点で処理を行なうため、効率が悪

The Implementation of Higher-order Function and Lazy Evaluation to Datarol Machine

Hisakazu KAWACHI, Eiichi TAKAHASHI, Rin-ichiro TANIGUCHI, Makoto AMAMIYA
Interdisciplinary Graduate School of Engineering Sciences,Kyushu University

い。Datarol モデルでは部分適用によって得られる新しい関数を結果として返すことはできないため、本方式では代わりにインスタンスのレジスタファイルを結果として返すことにより部分適用を行なう。しかし、Twice のように結果として返されるレジスタファイルを複数用いる場合があるため、レジスタファイルのコピーが必要となる。現アーキテクチャーでは、コピーを行なう機構がないため CU(Copy Unit) を実装しなければならない。

例 2.1

```
f(x) = lambda(y) x*x*x+y
F = { let s = f(2)
      in s(3) }
```

例 2.1 の f(x),F の Datarol グラフを図 3 に示す。

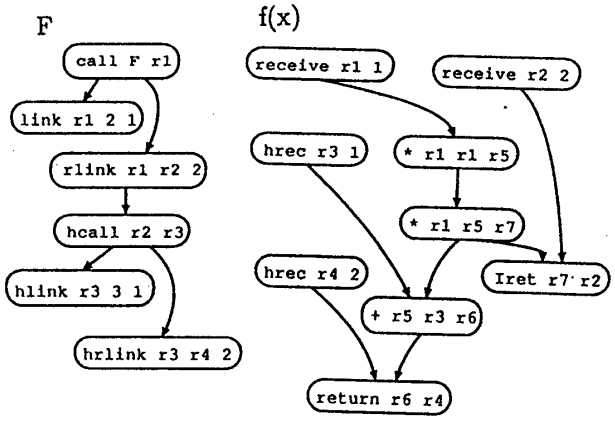

図 3: f(x),F の Datarol グラフ

Iret：インスタンス及び hreceive の LM アドレスを返す。

hcall：関数インスタンスにレジスタファイルを確保し、オペランドの示すインスタンスのレジスタファイルをコピーする。

hlink,hrlink：hreceive を叩く。

3 遅延評価の実装方式

3.1 記述方式

Valid において、評価の遅延と開始の指定は、それぞれ delay,force オペレータを用いて次の様に行なう。

```
{ let u=delay g(x,y)
      in f(u) }
  function f(u) = ...force u ....
```

3.2 実装方式

3.2.1 SM recipe 方式

delay 文に対して Datarol コンパイラは、2.2.1 と同様の recipe を構築するコードを出す。recipe に遅延すべき関数のコードと引数を蓄えておき、新しく導入した force 命令によって遅延した関数を起動する。

force：recipe 中の関数にレジスタファイルを確保し、関数の receive 命令に recipe 中の引数を渡す。

3.2.2 zcons の実装方式

遅延評価は、無限リストなどのリスト操作において有効である。しかしながら、3.2.1 の方式では、delay の度に SM のアレイ部に recipe を作り複雑な動作となる。このため、リストの遅延評価には zcons を用いることにする。例えば、n からの無限リストを生成する関数 from は、zcons を用いて以下のように Valid で記述する。

```
function from(n:integer) return(list)
  = zcons(n,from(n+1));
```

zcons を実装するにあたり、zcar,zcdr という新しい命令を導入する。from の Datarol グラフを図 4 に示す。

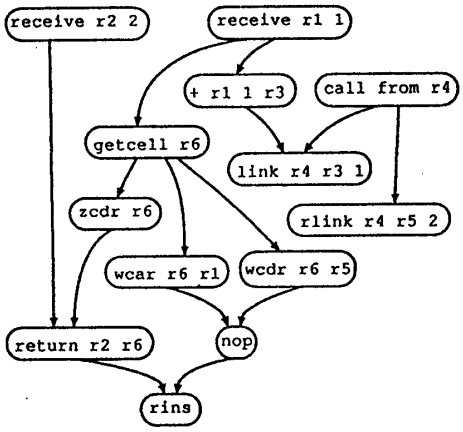

図 4: from(n) の Datarol グラフ

cons セルには、ready タグ（値が到着しているかどうか）と code タグ（LM アドレスかどうか）の 2 つのタグがある。from の場合 zcdr によって call の LM アドレスが cdr 部に書き込まれ、ready タグ、code タグをオンにする。この cdr 部を cdr 命令によってアクセスすると、まず、cdr 命令を pending する。次に ready タグをオフにして、call 命令を叩き、遅延していた cdr 部の計算を開始する。値が返ってきて wcdr を叩くと ready タグをオン、code タグをオフにし pending していた cdr 命令を再開する。

4 まとめ

本稿では、高階関数、遅延評価の Valid での記述法及び Datarol マシンへの実装方式について述べた。ここで述べた手法を実装することにより、Datarol マシンに要求駆動的動作を取り込むことができ、データ駆動方式では処理できない問題なども処理できる。また、マシンレベルで高階関数を実現でき、高速実行が可能となる。今後は、シミュレーションを進めていき、高階関数・遅延評価の最適な実装方式について考察を進めていく。

参考文献

[1] Amamiya,M. and Taniguchi.R : Datarol:A Massively Parallel Architecture for Functional Languages, Proc.SPDP, 1990, pp.726-735.

[2] 長谷川, 雨宮:データフローマシン用関数型高級言語 Valid, 電子情報通信学会論文誌 Vol.j71-D No.8(1988)