4G−6

関数的実行環境下における構造データ処理に関する考察

井上隆仁　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

数学的な関数の概念に基づく関数型言語は，プログラムに内在する並列性を利用して並列処理プログラムを容易に導出することが可能であり，大規模な並列処理を記述する上で有効である．

関数型言語の実行にあたり，関数概念に忠実に変数の破壊的代入を認めない環境を，関数的実行環境と呼ぶ．このような環境において，アレイやリストなどの構造データは，その要素が生成・変更される度に新しい構造データが確保されることになる．しかし，これはメモリ資源の有効利用という観点から見て現実的ではない．そこで筆者らは，確保する構造データを必要最小限に抑え，可能な限り破壊的代入を行なうオブジェクトコードを生成する方式を検討している．

本稿では，我々が開発している関数型言語 Valid [1] のコンパイラ[2] に本方式の導入を検討する．

2 並列式

並列式はベクトル演算のような並列実行可能な処理単位を記述する．値として，並列式本体で生成された値を要素とするタプル（後述）を返す．各々の処理は fork/join 概念に基づいて一斉に実行することができる．例えば，

for each i **in** [1..n] **do** f(i)

は次のタプルを生成する．

(f(1), f(2), ···, f(n))

3 構造データの処理

3.1 オペレータ

構造データを処理するためのオペレータを以下に示す．

1. **creat_vector**(n, type)
   大きさが n でデータの型が type であるベクトル領域を確保する．ヘッダにベクトル領域の大きさとタイプが入る．値としてベクトル領域の先頭アドレスを返す．

2. **make_vector**(x)
   タプル x からベクトルを生成する．

3. **make_list**(x)
   タプル x からリストを生成する．

4. **read**(x, i)
   ベクトル領域 x の第 i 要素を読み出す．x[i] と表現する．
   x[i, j, k] == x[i][j][k] == **read**(**read**(**read**(x, i), j), k)

5. **write**(x, {(i_1, v_1), (i_2, v_2), ···, (i_n, v_n)})
   既定義ベクトル x の第 i_k 要素の値を v_k としたベクトルを値とする($k = 1, 2, \cdots, n$)．{ } 内は，(i_k, v_k) のタプルである．

3.2 タプル

タプル (tuple) とは，データが横一列に並んだ状態にある仮想的なデータ構造である（*cf.* FP の sequence [3]）．具体的な構造はないが，データの位置関係だけは保証されている．

前述したように，並列式はタプルを値として返す．並列式を実行するときのイメージを図 1 に示す．

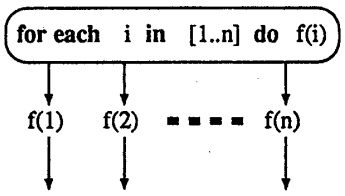

図 1: タプルの概念

make_vector は，タプルとして並列に流れてきたデータをベクトルとしてメモリに格納するためのオペレータである．データは，タプルのもつ位置関係が保持された状態で格納される（図 2）．

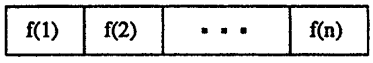

図 2: タプル → ベクトル

Structured-Data Processing Under Functional Execution Environment

Takahito INOUE, Rin-ichiro TANIGUCHI, Makoto AMAMIYA
Department of Information Systems, Graduate School of Engineering Sciences, Kyushu University

また make_list は，タプルとして並列に流れてきたデータからリストを生成するためのオペレータである．データは，タプルのもつ位置関係に従った順で格納される（図 3）．

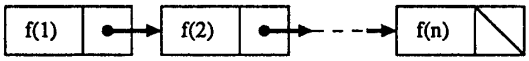

図 3: タプル → リスト

4 記述例と考察

例 1: n × n 行列の定義（ make_vector を用いた例）

```
A: 乗算表
   A = make_vector(for each i in [1..n] do
     make_vector(for each j in [1..n] do i*j))
B: A × A
   B = make_vector(for each i in [1..n] do
     (make_vector(for each j in [1..n] do
       for (k, s):(1, 0) do
           if k > n then s
           else recur(k+1, s+A[i, k]*A[k, j]))))
```

例 2: 1 次元配列の定義（ read,write を用いた例）

```
C: 1 次元配列
   C = make_vector(for each i in [1..n] do i*i)
D: C をシフト (C[i] → D[i+1])
   D = write(C, for each i in [1..n-1] do (i+1, C[i]))
```

write は，第 1 引数で与える既定義ベクトルの要素を変更したベクトルを返す．従って，新しいベクトル領域を確保しなければならない．しかし，要素の一部だけが異なるベクトル領域を確保するのはメモリ資源の無駄である．そこで，既定義ベクトルに対する参照が全て終了した時点で，既定義ベクトル領域に変更する要素を破壊的に書き込むようにする．この方式により，新しい領域の確保が不必要となりメモリの使用効率を上げることができる．

本方式を実現する上で，記述例 2 において以下の問題が生じる．

1. D は C の領域に対して破壊的書き込みを行なうことによって得られる．従って，破壊的書き込みを行なう以前に必ず C に対する他の参照が終了していなければならない．

2. D の定義の中には， C の read と write の両方が含まれている．従って，read を実行してから write を実行しないと意図したものと違う D が得られてしまう．

1 を解決するためには，構造データの依存解析が必要である．依存解析によって，他からの参照の終了を求める．依存解析のアルゴリズムは，一般の変数に対する依存解析と同様に構造データの名前に着目して行なえばよい．

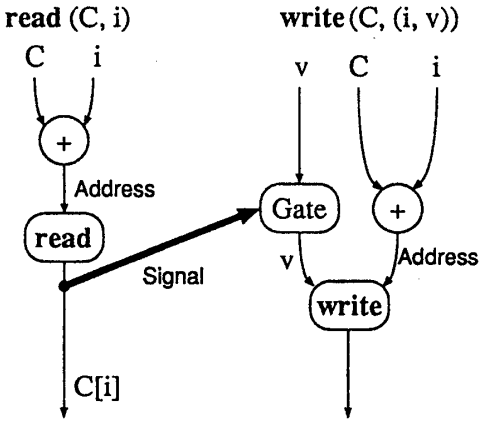

図 4: 同期制御

また 2 を解決するためには， read/write の同期制御が必要である．記述例 2 の第 i 要素に対する read と write に同期制御を行なうと，図 4 のようになる．

一般的には，どの read 命令からどの write 命令に同期制御を行なうためのシグナルを送る必要があるかということを決定しなければならない．そのために，一つの構造データ内の要素の依存関係を解析する必要がある．この依存解析アルゴリズムについては，動的にしか依存関係を解析できない場合があり，現在検討中である．

5 まとめ

純関数的実行環境下において構造データの処理を行なう際にタプルという概念を用いることによって，関数性を保証できることとを述べた．今後は，現在使用している Valid コンパイラに本稿で述べた手法を組み込み，オブジェクトコードの最適化に関する研究を進めていく予定である．

参考文献

[1] 雨宮真人，長谷川隆三: “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, pp.1532-1539 (1988).

[2] 立花徹，谷口倫一郎，雨宮真人: “データフロー解析による関数型言語 Valid のコンパイル法 — Datarol プログラムの抽出アルゴリズム —”，情報処理学会論文誌，Vol.30, No.12, pp.1628-1638 (1989).

[3] J. Backus: “Can Program be Liberated from the Von Neumann Style? A Function Style and its Algebra of Programs”, Commun. ACM, pp.613-641 (1978).