5B-2

データフローマシン向きLisp型言語に関する一考察

松方　純・田中　英彦・元岡　達
(東京大学　工学部)

はじめに

データフローマシン向き言語として関数型言語が有力である。Lispも関数型言語の一つであるが、逐次処理を前提として発展してきたため、データフローマシンの特長である並列処理を行ないにくく、かならずしもデータフローマシン向きではない。本稿では、Lispをデータフローマシン向きに修正したデータフローマシン向きLisp型言語について考察する。

Lispのプログラムのデータ駆動的実行

データフローマシン(データ駆動型)のプログラムは、データフローグラフで表現される。Lispのプログラムをデータ駆動的に実行するには、プログラムをデータフローグラフに変換する必要がある。図1は、次のようなLispのプログラムをデータフローグラフに変換したものである。

```
(LABEL FACT
  (LAMBDA (N)
    (COND ((ZEROP N) 1)
          (T (MUL (FACT (SUB1 N)) N)))))
```

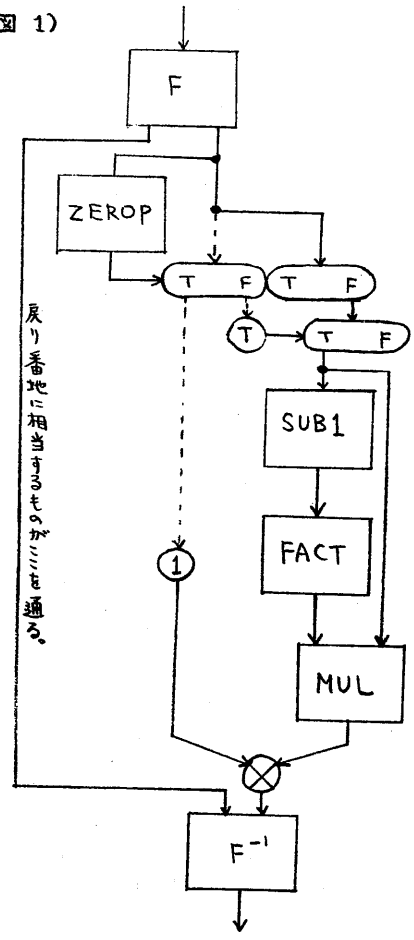
このような変換を行なうには、Lispのプログラムが次の条件を満たすことが望ましい。

(1) 関数に副作用がないこと。
(2) 関数に自由変数がないこと。
(3) 変数のscopeがstaticに定まること。
(4) 関数の引数の評価の有無が容易にわかること。
(call-by-value[1]か、call-by-name[1]か。)

(1)と(2)は、データフローグラフのノードのもつ性質であり、したがって、関数とノードの対応を一対一にするための条件である。

(3)と(4)は、Lispをコンパイルする際に望ましい条件である。(3)が成立しない場合、すなわち、変数のscopeがstaticに定まらない場合には、binding mechanismが必要となり、処理が複雑になる。なお、

(図1)



F：関数入口演算子
トークンのタグをローカルなものにつけかえる。
F^{-1}：関数出口演算子。
↓：トリガ(値のないトークン)をあらわす。

(2)(関数に自由変数がない。)の条件が成立するとき、変数のscopeは、staticに定まる。すなわち、(3)も成立する。

(4)については、普通、関数の定義を参照すれば、引数の評価の有無がわかるので、普通のLispはこの条件を満たす。しかしながら、この条件をもう少しきびしく解釈して、プログラムの字面のみからcall-by-valueかcall-by-nameかわからなければいけないものとすると、大部分のLispは、この条件を満たさない。

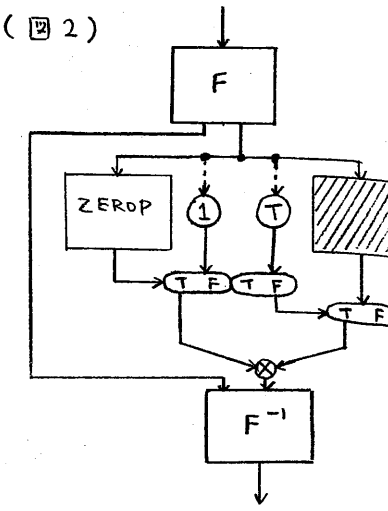
関数の引数の並列処理

前節で述べた変換を用いると、関数の引数の並列処理は、自然に導入される。関数のうち、condのように引数の並列処理が難しいものもある。

condの並列処理

condの並列処理とは、condの引数の並列処理のことである。図1では、condは並列処理されてない。図2は、condを並列に処理するようにしたものである。図2においても、図1と同様に、有限の時間内に正しい結果が出力される。しかしながら、▨内での計算は、▨が結果を返したあとも際限なく続く。このように、condの並列処理には解決すべき問題が多い。

(図2)



(注)
(MUL (FACT (SUB1 N)) N)に相当する部分を▨と略記した。

おわりに

以上のように、Lispのプログラムをデータフローグラフに変換することを中心に、データフローマシン向きLisp型言語について考えてみた。Lispのプログラムをデータフローマシンで処理する際問題になることは、他にもいろいろあるたとえば、S式を、数値のようにトークンとして流すのか、何か別の方法を使うのかという問題がある。Lispの記号処理言語という側面を考えたとき重要な問題である。

参考文献

1) John Allen : Anatomy of Lisp , McGraw-Hill (1978)