6Q-4

並列オブジェクト指向言語 FLENG++ の実装

中村 宏明, 田中 英彦
(東京大学 工学部)

1 はじめに

FLENG++ は, 大規模な知識処理を行なう並列計算機 [1] でのプログラミングを容易に行なうことを目的として設計されたオブジェクト指向言語である. FLENG++ のプログラムは, すべて GHC を単純にした committed-choice 型言語 FLENG[2] にコンパイルの後, 実行される. 今回は, FLENG++ とその処理系, 開発支援環境について報告する.

2 FLENG++ の概要

2.1 クラス定義とオブジェクト

FLENG++ ではクラスを定義することによってプログラムを作成する. クラス定義は, クラス・ヘッダと述語定義部から構成される. クラス・ヘッダにはクラス名, インヘリタンス, インスタンス変数名等が記述され, 述語定義部はオブジェクト間でやりとりされるメッセージに対応するメソッド定義と, 必要に応じてオブジェクト内での処理に用いられる FLENG のローカルな述語定義から構成される. このクラス定義にしたがってオブジェクトが生成されるが, これらクラス定義から生成されるオブジェクトの他にアトムや数字などのリテラル表現が可能なプリミティブなオブジェクトがある.

2.2 メソッド定義

メソッド定義は FLENG の述語定義を拡張したもので, 以下のように行なう.

```
:<メソッド名>(<オブジェクト>,<引数 1>,...,<引数 n>) :-
    <ゴール 1>,...,<ゴール n>.
```

述語名に ':' を前置したものはメソッド名とみなされる. また, ヘッドの第一引き数ではオブジェクトが受け取られる. ゴールはメソッド呼び出し, または FLENG 述語呼び出しである. このメソッド定義は ESP[3] の記法にしたがってものであるが, これは以下の理由による.

- FLENG++ のメソッドと FLENG の述語定義, 呼び出しを自然に混在させて記述することができる.
- 擬変数などを用いた特別な扱いをせずに自分自身を参照できる. このため以下で述べるプログラムの解析, 最適化が容易になる.
- 我々は今までに, ESP でのプログラム作成技術や, プログラミング環境の研究を進めてきたが, これらの蓄積を利用することができる.

ただし ESP と異なり, FLENG++ では並列計算のモデルとして単純である必要があることと, 並列システムでは集中管理が困難なことから, クラスはオブジェクトではない.

2.3 変数

変数には 2 種類のものがある. 1 つはインスタンス変数で, そのスコープはオブジェクトが生成されてから消滅するまでであり, 値の書換えが任意に行える. もう一つはテンポラリ変数で, メッセージ受信の度に新しく生成される単一代入の変数である. この 2 種類の変数を使い分けることによって, 並列性を阻害せず, 柔軟なプログラミングが可能になる.

2.4 プログラム例

```
class ex                                          (1)
  var $list := [a].                               (2)
:add(_,L) :-
    append($list,L,NewL),$list := NewL.           (3)
:get(_,!$list).                                   (4)
append([],Y,!Y).                                  (5)
append([A|X],Y,![A|Z]) :- append(X,Y,Z).          (6)
end.                                              (7)

?- :new(#ex,E),:add(E,[b]),:get(E,L).             (8)

L = [a,b]
```

(1), (2) はクラス・ヘッダである. インスタンス変数には初期値を代入しておくことができる. (3), (4) はメソッド定義である. 各述語呼び出しは並列に実行することができるが, (3) や (8) において, インスタンス変数への代入, オブジェクトへのメッセージ送信は副作用を生じ, 実行順序によって結果が異なる. このような場合, 左にあるゴールから実行されるようにコンパイルを行なう. (5), (6) はローカルな FLENG の述語定義である. クラス定義の外に記述されると, その定義はグローバルなものになる. ヘッドの引数に付加されている '!' は出力を示すアノテーションである. これによってプログラムが簡潔になると同時に, 処理系に対して最適化のヒントを与える.

3 FLENG++ の処理系

FLENG++ のプログラムは FLENG に変換して実行するのであるが, 両者の間にはかなり大きなセマンティック・ギャップがあるので, コンパイルのためにはプログラム解析等のやや複雑な処理が必要になる. ここではオブジェクトに対するリファレンスが増加する際の処理とインヘリタンスの扱いに絞って説明を行なう.

Implementation of Parallel Object-Oriented Language FLENG++
NAKAMURA Hiroaki, TANAKA Hidehiko
The University of Tokyo

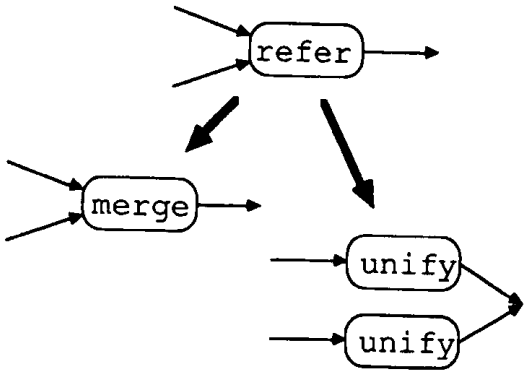

図 1: リファレンスを増加させる述語 refer/3

3.1 リファレンスの増加に対する処理

FLENG で状態を持つオブジェクトを実現するとき，メッセージ送信用の入力ストリームがそのオブジェクトへのリファレンスを示している．したがってリファレンスが増加するときには複数のストリームを併合する必要がある．一方，整数などのプリミティブなオブジェクトでは，リファレンスの増加は FLENG のレベルでユニフィケーションを行なうことによって行なう．FLENG++ の変数には型がないので，処理系が両者を区別して扱わなければならない．そこでリファレンスを増加するときには，図 1 のように merge/3 と unify/2 を行なう前に型のテストを行なう述語 refer/3 を挿入する．ここで，プリミティブ・オブジェクトの場合その項自身によって型を判別できるが，状態を持つオブジェクトでは通常のリストによるストリームの実現ではメッセージを待続ける (論理) 変数のままであるため refer/3 は両者を判別できない．そこで以下のような $s-list 構造によってストリームを表現し，msg1, msg2 の具体化に先行して常に複合項になるようにしておくことでストリームであることの判別を行なう．このため merge/3 の定義は通常知られたものと少し異なるが，詳細は省略する．

```
$s(msg1,$s(msg2,...))
```

また，ストリームには方向があるので，refer/3 の挿入には入出力モードがわかっていなければならない．これにはメソッド定義ヘッド部のアノテーション '!' と併用したコンパイル前段のモード解析を行なうことによって決定する．コンパイル時にモードが定まらないときは実行時に調べるコードを生成する。これはメッセージの送信に先だってストリームの枠組をつくっておくことによって可能である。

ところで，型のチェックを実行時に行なうのは効率の点で問題が多い．型をコンパイル時に決定するために，ESP のプログラミング環境の一部として開発したタイプチェック・推論アルゴリズム [4] を適用する。これによって型の定まらないものだけ refer/3 によって実行時に型をチェックするコードを生成する．型のチェックは，コンパイル時の最適化とバグ検出の両方に用いられる．

以上の様な機能によって、プログラマに負担をかけることなく無駄な処理を取り除いた実行が可能になる。

3.2 インヘリタンス

FLENG++ ではインヘリタンスは多重であり，コンパイル時に線形化しておく．知識表現を自然に行なうことを目的として，サブクラスがスーパークラスよりも先にたどられるように (depth first up to join) 順序付けを行なうが，ESP との交換性のため，指示によって深さ優先でコンパイルすることもできる．実行時にメソッドが適応可能かどうかを各クラスで順番に調べていく機構は，プロトタイピングの段階では柔軟な機構であるが，プログラムが完成した後ではオーバーヘッドになってしまう．そこで余分なメッセージの転送を省くために，受信する可能性のあるメッセージと各クラスでのメソッド名を調べることによってスーパークラスのメソッドを自クラスに融合するツールを用意する．このとき，メソッドのコピーを行なうためにコード量は増加する．つまり，実行速度とコード量がトレード・オフの関係にある．このため，クラスの継承関係等を表示させることによって，この決定をインタラクティブに行なうことも可能である．

4 FLENG++ に対応した FLENG の最適化

FLENG++ で記述されたプログラムの中には FLENG 最適化のためのさまざまな情報が含まれており、これを利用することによって FLENG の実行効率向上をはかることができる。一つは、FLENG++ でのメッセージの受信は FLENG においては構造データの待ち合わせとして表現されるが、これは常に単一参照になっているので、この領域を即時回収して新たな構造データを生成するときに再利用できることである。このために FLENG に対して単一参照アノテーションを付加する方式について検討を進めている [5]。

また、FLENG++ のメソッド検索は FLENG ではヘッドのユニフィケーションで実現されるが、オブジェクト指向言語ではメソッドの検索は特に高速であることが要求される。そこで通常のヘッド・ユニフィケーションに対してメソッド検索のためのハッシュテーブルを作成しておくことによって優先的に調べる。この表の作成のために、FLENG++ コンパイラは FLENG の処理系 [6] に直接アクセスする必要がある。

参考文献

[1] Koike H. and Tanaka H., "Multi-Context Processing and Data Balancing Mechanism of The Parallel Inference Machine PIE64", Proc. of FGCS'88, ICOT, 1988.

[2] Nilsson M. and Tanaka H., "FLENG Prolog - The Language which turns supercomputers into Prolog machines" In Proc. Logic Programing Conference. 1986, pp.209-216.

[3] Chikayama T., "ESP Reference Manual", Tech. Report, ICOT, TR-044, 1984.

[4] 小中他,"オブジェクト指向言語 ESP におけるタイプチェックシステムの試作", 情報処理学会第 36 回全国大会, 2H-5, 1988.

[5] 小池他,"単一参照アノテーションを用いた論理型言語の最適化コンパイル", 本大会.

[6] 下山他,"FLENG コンパイラとその抽象化コード", 本大会.