2Y-3

Committed-choice 型言語 FLENG と その上のユーザ言語 FLENG++

中村 宏明, 小中 裕喜, 田中 英彦
東京大学工学部

1 はじめに

我々は，大規模な知識処理を行うためK，高並列推論エンジンPIE[1]の開発を進めている． PIEの対象とする言語は，GHCKおける言語要素を簡単なものKし，並列計算機への実装を容易Kしたcommitted-choice型言語FLENG[2] である． FLENGの記述力を確かめるためK，FLENGKよるプログラミングの経験を重ねているが，簡単な言語要素であるKも関わらず，さまざまな並列アルゴリズムが記述できることがわかった．しかし以下のような問題点も見いだされた．

- 入出力のための機構が不十分で，特Kインタラクティブなプログラムの記述が難しい．
- 言語処理系やシステムの記述K不可欠な，プログラムをデータとして扱う機能K欠ける．
- PrologやGHCと比べると，同様な処理を行うのK，より多くの述語が必要Kなり，プログラムの可読性，保守性がよくない

我々はこれらの問題のなかでプログラムの可読性，保守性の問題を解決するため， FLENGの上位言語として並列オブジェクト指向言語FLENG++を設定し、今回はそのFLENG++ Kついて説明を行う．

2 committed-choice 型言語 FLENG

ゴールの成功，失敗を扱うためKはゴール生成の履歴を管理することや失敗を他のゴールK知らせることが必要であるが，並列計算機での実行を考えると，これは高価な処理である．FLENGは， GHC K対して以下のような変更を行うことKより，この問題を解消する．

FLENGではゴールの実行は論理的な真理値とは無関係K行われる．つまり他の論理型言語の意味で1つのゴールが失敗しても，他のゴールは影響を受けず，無関係K処理が進められる．ゴール間の論理的な関係が必要なときは，共有変数Kよってその関係を記述することが可能である．

ゴールが論理的な値を持たないとすると，GHCのガード部K当たるものを書くことができなくなる．FLENGではヘッドのマッチングが完了するとそこでクローズのコミットが行われる．ガード部はコミットの前K引数のテストを行うのK用いられるが，引数のテストはゴールの呼び出しの前K行うことができる．したがって次のような書換えKよってガード部を取り除くことが可能である．

```
(P1)    a(N) :- N < 0  | b(N).
        a(N) :- N >= 0 | c(N).
```

というGHCのプログラムは，

```
(P2)    a(N) :-
            compute(<,N,0,Result), a1(Result,N).
        a1(true,N) :- b(N).
        a1(false,N) :- c(N).
```

というFLENGのプログラムK変換できる．

ここで compute(Op,X,Y,Result)は FLENGの組み込み述語で， Opが比較演算子の場合、その結果K応じてResultがtrueまたはfalseK bind される．

3 上位言語 FLENG++

GHCではヘッドのマッチングとガードの条件Kよってクローズが選択されるが， FLENGではヘッドのマッチングK加えて，クローズの選択のための新たな述語が必要Kなる．このため記述量は相当増えることKなる． (P2)のプログラムは，呼び出す側で

```
...compute(<,N,0,Result), a1(Result,N)...
```

というようKプログラムを展開し，クローズの選択のための条件を計算するようKすればクローズの数は減らせる。しかし、このためプログラムのモジュラリティはきわめて低下する．

ここで我々は， FLENGの上位言語としてユーザがプログラムをつくるための言語，並列オブジェクト指向言語FLENG++を設定する．

オブジェクト指向の枠組みは，問題をプログラムK自然K反映できること，プログラムの共用・再利用が支援されるなどの理由Kより，さまざまな言語から積極的Kとりいられている．また，committed-choice型言語では特別な言語要素を導入しなくてもオブジェクト指向プログラミングが可能であることが知られている．

FLENG++のプログラムはFLENG Kコンパイルされて FLENGの処理系で実行される．

FLENG++の特徴は，実装の道具としてcommitted-choice型言語を用いるが，プログラムをつくるユーザKは，それを不必要K意識させないことである．たとえば，オブジェクトの

FLENG++ based on committed-chioce language FLENG,
NAKAMURA Hiroaki, KONAKA Hiroki, TANAKA Hidehiko,
The University of Tokyo

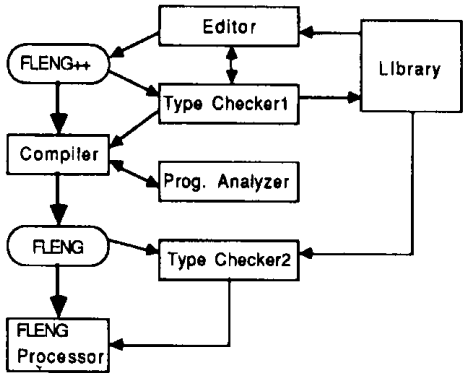

図 1: 図 1:FLENG++ システムの構成

内部表現は，アトムや数など primitive なものはユニフィケーションによってアクセスされる項であるが，ユーザの定義したオブジェクトは，そのオブジェクトをあらわす perpetual process への入力ストリームが merger を通してアクセスされることになる．このため両者の処理の方法はまったく異なるが，ユーザの側から見るとどちらも同じようにメッセージを受け取ることによって能動化されるオブジェクトである．また，次のようなインスタンス変数 slot への操作は

```
slot := slot + 1,
<slot を使った操作>,
slot := 0,
```

次のように FLENG のプログラムへ変換される．

```
plus(Slot0,1,Slot1),
'$usage'(Slot1),
unify(_,Slot2,0),
```

つまり，異なるオブジェクトへのメッセージ送信は並列に処理されるが，同じオブジェクトへ複数のメッセージを送ったときも，論理変数の依存関係によってデータフロー的に処理が並列に行われる．ユーザはこのことを意識することなく逐次的にプログラムを書いてよい．

ユーザがプログラムを書きやすくなる一方で，FLENG++ のプログラムから FLENG のプログラムへのコンパイルは複雑になる．また，本当にプログラムをつくりやすくするためには，プログラミング環境の整備が重要である．FLENG++ の処理系は図 1 のようになっている．コンパイラはタイプ・チェック[3] とモード解析の結果を用いてプログラムのコンパイルをすすめる．また，タイプ・システムはコンパイル時に起動されるだけでなく，プログラムをエディタから入力しているときなどにライブラリを参照しながら誤りを発見しユーザに報告を行うなどの目的にも用いられる．また高級言語へのコンパイルでは処理速度が問題になるが，一般にコンパイル前のソース・レベルでその性能を予測することは難しい．このため，コンパイル時に静的な特性を収集し，また実行時に動的な特性解析を行うためのコードを生成することによりプログラムの特性を測定し，プログラムの改良に役立てる．

4 おわりに

現在，上記の FLENG++ システムの構築をすすめている．さらにプログラミング環境を充実させるために効率のよいデバッガやクラスの管理を行なうライブラリの検討を行なっている．また，現在の FLENG++ システムは Prolog で記述しているが，FLENG にメタ・レベルの機能を充実させ，FLENG によってすべてのシステムの構築を行なえるようにしたい．

参考文献

[1] 小池, 田中,“並列推論エンジン PIE の全体構成”, 本大会
[2] Nilsson M. and Tanaka H., “FLENG Prolog - The Language which turns supercomputers into Prolog machines Logic Programing Conference. 1986, pp.209-216.
[3] 小中, 田中,“ESP のタイプシステムの実装と並列言語への展開”, 本大会