TAO/ELIS 上での並行論理型言語の実現

4Q-7

増尾 剛
NTT
NTT ソフトウェア研究所

1 はじめに

TAO[1]は、Lisp マシン ELIS 上におけるマルチパラダイムプログラミング言語である。一般の Lisp における関数型パラダイムの他に、論理型パラダイム、オブジェクト指向パラダイムを兼ねそなえている。また、TAO はマルチプロセスが可能で、そのための様々な基本機能が用意されている[2]。しかし、これらの機能は若干プリミティブで、必ずしも一般のプログラマにとって使いやすいものとはいえない。そこで、TAO のマルチプロセス機能をプログラマにとって利用しやすいものとするため、TAO 上に並行論理型言語 CLL を実現した。CLL は TAO のマルチプロセス機能を、TAO の 3 つのパラダイムのうち特にそのセマンティクスが並行実行と相性が良い論理型パラダイムと組み合わせることによって、抽象化したものである。

2 並行論理型言語 CLL

CLL は一般の DEC-10 準拠の Prolog のシンタックスに、並行処理のためのマルチプロセスとプロセス間通信の機能を付加したものであり、GHC などの Comitted-Choice 言語よりも、Delta-Prolog[3]の系統に属するものである。マルチプロセス機能は次のような記法によって明示的に表される。

- `,` ゴールの逐次 AND 結合。つまり、一般の Prolog の , オペレータと同じである。
- `//` ゴールの並行 AND 結合。

```
:- (P1 // P2), (P3 // P4).
```

は、まず P1 と P2 のゴール呼び出しを並行に行い、次に P3 と P4 のゴール呼び出しを並行に行う。このとき、例えば、P1 // P2 で、P1 が P2 より先に失敗終了しても、ただちに P1 // P2 を強制的に失敗終了させることはせずに、P2 の終了を待ち合わせる。これは、強制終了させてしまうと、通信が失われ、デッドロックを引き起す可能性があるからである。

プロセス間通信は共有変数ではなく、チャネルを介したメッセージ通信によって行われる。具体的には、チャネルを介して項同士を unification することによって通信が行われる。これらは以下に示すような infix operator として表される。

- `T1 $ Ch`
  チャネル Ch に対して項 T1 を非同期的に送信する。つまり、これを実行したプロセスは中断することなく、次の実行に進む。このとき T1 は基底項でなければならない。この述語は必ず成功する。
- `T2 ? Ch`
  チャネル Ch から項 T2 に受信する。すでに他のプロセスから Ch に送信、つまり例えば T1 $ Ch が実行されていれば、T2 と T1 との間で unification が行われる。まだ、送信が実行されていなければ、実行されるまで suspend する。unification が失敗した場合、この述語 T2 ? Ch の失敗となり、? を実行したプロセス内で backtrack が引き起される。
- `T2 ? Ch:Cond`
  チャネル Ch から項 T2 に条件付きで受信する。通信が成立し、送信側の項と T2 の unification が成功した後で、Cond が実行される。Cond の実行までが成功すると T2 ? Ch:Cond の成功になる。Cond が失敗すると上で述べた unification が失敗したときと同様に、T2 ? Ch:Cond の失敗となり、これを実行したプロセス内で backtrack が引き起される。
- `T3 ! Ch`
  チャネル Ch に対して項 T3 を同期的に送信する。Ch に対して既に他のプロセスが受信、例えば、T2 ? Ch が実行されていれば、T2 と T3 の間で unification が行われる。Ch に対して ? がまだ実行されていなければ、実行されるまで T3 ! Ch を実行したプロセスは suspend する。unification が失敗した場合、T3 ! Ch は次に Ch に対して ? が実行されるまで suspend する。
- `makeChan(Ch)`
  チャネルを生成する組み込み述語である。チャネルを生成し、Ch に bind する。

CLL に OR 並列は存在しないが、Committed-Choice 言語のコミット機構に相当する並列探索は可能である。これは以下のような宣言によって行われる。

```
:- parallel Pred
```

この宣言により、述語 Pred に対するゴール呼び出しは並列に行われる。ただし、Pred の body 部の第一ゴールは必ず、メッセージ受信 ? でなければならない。これは、Committed-Choice 言語における guard 部に相当する。メッセージが受信され、受信側の項との unification が成立し、そしてもしあれば、条件文も成功した初めての節が一つだけ選択され、残りの節は棄却される。

Concurrent Logic Language on TAO/ELIS
Tsuyoshi MASUO
NTT Software Laboratories

3 プログラム例

文献[3]のsquareプログラムと、counterプログラムのCLLによる例を次に示す。

```
squares(Ch) :- write(0), nl, sq(0,Ch)
sq(Q,Ch) :- I ? Ch,
            R is Q + I, write(R), nl,
            sq(R,Ch).

odds(Ch) :- odd(1,Ch).
odd(I,Ch) :- I $ Ch, J is I + 2,
             odd(J,Ch).

go :- makeChan(Ch), (squares(Ch) // odds(Ch)).
```

プログラム (1) square プログラム

```
:- parallel c/2.

terminal(Ch) :- read(Com), Com ! Ch,
                write(Com), nl, terminal(Ch).

c(S,Ch) :- clear ? Ch, c(0,Ch).
c(S,Ch) :- up ? Ch, U is S + 1, c(U,Ch).
c(S,Ch) :- down ? Ch, D is S - 1, c(D,Ch).
c(S,Ch) :- show(S) ? Ch, c(S,Ch).
c(S,Ch) :- abolish ? Ch.
c(S,Ch) :- X ? Ch:other(X), c(S,Ch).

go(Init) :- makeChan(Ch),
            (terminal(Ch) // c(Init,Ch)).
```

プログラム (2) counter プログラム

プログラム (1) は、一種の生産者・消費者問題である。生産者プロセス (odd) と消費者プロセス (sq) が通信し合いながら、並行に実行される。

プログラム (2) は、CLLによるオブジェクト表現である。内部状態を持つオブジェクト (c) が、外界 (terminal) とのチャネルを通じて、自分の内部状態の報告や変更を行う。

4 TAO/ELIS 上における実現

TAOには一般のDEC-10 Prologシンタックスで書かれたプログラムをそのままの形で読み込んで実行できるパッケージが存在する。それにCLL独自のプロセス生成やチャネル通信用の機能を組み込み述語の形でパッケージシステムに付け加えることにより実現した。この方法によって、CLLは完全にDEC-10 Prologのスーパーセットになり、普通のDEC-10 Prologシンタックスで書かれたプログラムもそのままCLL上で実行できる。

チャネルによる通信はTAOのmailboxによって実現される。実際、CLLのチャネルはまさにTAOのmailboxそのものである。TAOにはmailboxに対する機能 (関数) として、非同期送信のsend-mailや受信のreceive-mailがあるが、同期送信の機能はない。このため、CLLにおける同期送信は、システムの内部で (TAOで) 応答用のmailboxを用意することにより実現されている。

並行AND `P1 // P2` は、関数process-forkによって実現されている。forkされた`P1, P2`は、TAOの論理型パラダイムによって実行される。論理的なAND関係を実現するため、process-forkの実行のさいに実行結果通信用のmailboxが用意される。`P1, P2`は実行が終了すると、その結果 (success/failure) を通信し、その組み合わせによって、`P1 // P2`の結果 (success/failure) が決定される。

`:- parallel` 宣言による並列探索は、読み込み時にプログラムを内部的に変換することによって実現される。プログラム (2) のように、一つのチャネルからの受信パターンによって、節が一意に決まる述語に対する宣言では、各節は内部的にはそれぞれ違う述語名で読み込まれ、単独の節として認識される。そして、その述語を呼び出すときはそれぞれの内部的に異なる述語名ごとに対して、ゴール呼び出しプロセスをforkする。forkした各プロセスは、第一ゴールで受信待ちになり、受信パターン、および条件文が成功したプロセス (これは一つだけである) が実行を継続し、その他の失敗したプロセスは消滅する。

複数のチャネルから非決定的に待っているような節にたいしては、TAOのmultiple-waitという、複数のイベントを並列的に待つことのできる特殊形式マクロに変換する。

5 今後の課題

今後、実際にCLLで中規模程度のアプリケーションを書いてみて、並行プログラミングに必要な機能の検証を行う予定である。特に、並行プログラミングにおけるデバッグ手法について、通信を抽象化したことによる有効性を検討する。さらに、ネットワークによる分散環境でのCLLの実現も検討していきたい。

参 考 文 献

1) Takeuchi, I., Okuno, H. and Osato, N.
A List Processing Language TAO with Multiple Programming Paradimgs.
New Generation Computing No.4 1986.

2) Takeuchi, I.
Concurrent Programming in TAO - Practice and Experience.
Proc. of US/Japan Workshop on Parallel Lisp,1989.

3) Pereira, L.M. and Nasr, R.
DELTA-PROLOG: A Distributed Logic Programming Language.
Proc. of the International Conference on Fifth Generation Computer System, 1984.