データフローに基づくマルチエージェントによる並行プログラミング

7M−7

友清孝志　日下部茂　谷口倫一郎　雨宮真人
九州大学大学院総合理工学研究科

1. まえがき

問題解決などの処理には，多数の自律的な処理体が通信しあいながら処理を行う並列協調システムが有望視されているが，そのようなシステムの実現には大規模な並列処理が不可欠となる．我々は超並列分散処理にはデータフロー計算モデルが適しているとの立場から，これに基づく並列協調システムの具体化を進めている [4]．

並列協調システムの単位となる処理体は，それぞれ固有の状態値を持ち，その状態値を参照しながら履歴に依存した処理を行う必要がある．データフロー概念に元来存在しない状態の概念を扱うために，我々はデータ駆動原理に基づいて動作し，しかも状態をもつ協調処理単位であるモジュールを導入してこれに対応し，併せて関数型言語 Valid を拡張した [5]．

一般に関数性とオブジェクト指向の考えは相反する概念として認識されているが，我々のモデルは再帰とストリーム概念に基くことにより，代入による副作用をおこすことなく両者の利点を融合している．

本稿では，並行プロセスにおける諸問題を例にとって，Valid によるプログラミングを示し，本モデルの有効性を示す．

2. モジュールの構造

モジュールは動的に生成，消去され，自律的に並行動作する処理体であり，内部に持つ複数の関数を，受理メッセージとのパタン照合により選択的に起動する．それらの関数はモジュール内部の状態値を参照しながら履歴に依存した処理を，データ駆動原理に基づいて行う．その履歴情報は状態値を陽にフィードバックすることにより保持される．この自己へのフィードバックを一種のストリームとして考えれば，システムに状態がないと見なすことができる．この点では代入に基づく他のモデルとは異なり，関数性を保っている．

複数個のモジュールは互いにバッファつき非同期通信により交信しあう．また応答メッセージとの同期を陽にとることもでき，この機構もデータ駆動により行われるため，モジュール内とモジュール間の両者においてデータ依存関係による最大限の並列性が自動的に抽出される．なおメッセージはシステムによりキューとして用意され，状態がフィードバックされる度に先頭より取り出される [5]．

実際の処理の様子は次節で具体的に例示する．

3. 並行プロセスの諸問題の記述

並行処理プログラムにおいては様々な資源競合の問題が発生する [1]．以下では，まず銀行口座のプログラムを例にとり Valid におけるモジュールの記述法を説明し，相互排除問題，哲学者の食事問題，読み出し書き込み問題について考察する．

3.1 銀行口座

Valid による銀行口座のプログラムを図 1 に示す．

```
module Bank_account (init_balance)
={ for (balance):(init_balance) do
    recur(new_balance)
where
  mess=head(message_queue()),
  new_balance = case
    mess = ['withdrawal|?n] →
      if balance ≥ n then
        {balance-n where !=reply('done)}
      else
        {balance where !=reply('overdraft)},
    mess = ['deposit|?n] →
      {balance + n where !=reply('done)},
    mess = ['show_balance|_] →
      {balance where !=reply('balance)}
  end },
```

図 1. 銀行口座のプログラム

まず変数 mess がメッセージストリームの先頭要素に束縛される．モジュール本体は末尾再帰型の関数であり，状態値 balance を再帰的に更新していく．更新処理はパタン照合により起動アクションが選択され，その関数値が次の状態値となる．また同時にメッセージの送り主に対して返答メッセージが返される．

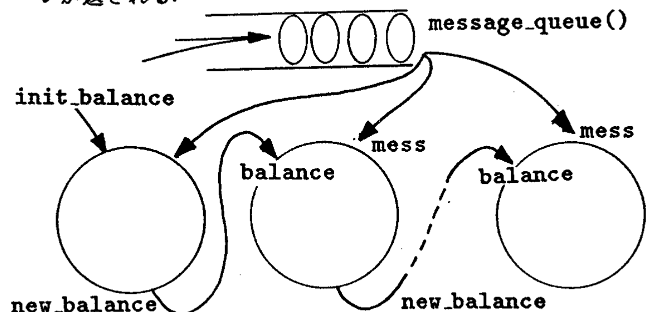

図 2. 図 1 のプログラムの実行の様子

このプログラムの実行時の様子を図 2 に示す．このように再帰式は展開 (unfolding) され，各ループフェーズ間の同期もデータ駆動により行われる．

さて，代入により局所状態を変更する一般のモデルでは，状態の更新時に相互排除を行う必要がある．このことは並列オブジェクト指向モデルにおいても例外ではなく，そのため他のモデルでは複数メッセージの並行処理をする際，セマフォなどの機構を用いるか，またはこのような処理をあきらめるかになっている．

しかし我々のモデルでは，上図のように前回の状態の更新が次の更新処理の発火の為に必要であり，この同期処理はデータ駆動により自動的に行われる．したがってこの例の場合，複数の異なる支店からの送金を並行して処理することができる．

Concurrent Programming in dataflow-based multi-agent system

Takashi TOMOKIYO, Shigeru KUSAKABE, Rin-ichiro TANIGUCHI and Makoto AMAMIYA
Graduate School of Engineering Sciences, Kyushu University

3.2 哲学者の食事問題

哲学者の食事問題はデッドロックをモデルした古典的な問題であるが，我々のモデルにおいてはデッドロックは本質的な問題ではない．まず，要求する資源が相互排除を要する共有資源であることは，デッドロックが発生するための条件の一つであるが，モジュール内にカプセル化された状態値は，ストリームとしての時系列データであり，これ自体は共有されていない．またデッドロックはそれ以上処理を先へ進ませることができない状態であるが，モジュール間で待ち合わせのループが起こった場合でも，次のループインスタンスが処理を進めることが可能である．

しかし，物理的な資源に関しては，依然としてデッドロックは起こり得る．ここでは，箸を2本得ることができないならばもとに戻す，という一般的な解を示す．

```
module philosopher(右の箸,左の箸)
={ for (state):('thinking) do recur (new_state)
where
  new_state = case
    state = 'thinking →
      { 'hungry after s
        where s=wait(random()*20) }
    state = 'hungry →
      {let x = send(右の箸,'pick),
           y = send(左の箸,'pick)
       in if x='free and y='free then 'eating
         else { 'hungry
              where !=send(右の箸,'put),
                    !=send(左の箸,'put) }
    state = 'eating →
      { 'thinking after s
        where s=wait(random()),
          !=send(右の箸,'put),
          !=send(左の箸,'put) }
  end },
```

図3. 哲学者の食事のプログラム

箸自身もモジュールとすることにより，send文に対して，箸は自分の状態値を変更すると同時に以前の状態値を返す．これにより，Test & Setのように不可分な処理としてメッセージは処理される．またここで，右の箸への送信と左の箸への送信は並行に行なわれる．

3.3 読みだし書き込み問題

読みだし書き込み問題[1]のプログラムを図4に示す．この問題は，書き込み者優先で複数の読みだし者と書き込み者に対して排他制御を行なうものである．銀行口座の例で示したとおり，モジュールの局所状態としてモデル化したものへのアクセスの場合は，複数のアクセスは問題なく処理される．この問題の場合も物理的な資源を管理する場合にのみ生じてくる．ここでは，ファイルマネージャとしてプログラミングを行なった．このモジュールは，後回しにするメッセージを保留しておくためのキューと，現在読みだし，書き込みをそれぞれ行なっている者のカウンタとを状態値として保持し，読みだし/書き込み要求とそれぞれの完了シグナルを受け付ける．まずメッセージを先読みして，完了シグナルであればカウンタを減じ，要求メッセージであれば内部キューへ付け加える．そしてキューの先頭とカウンタの状態によりアクションを選択的に起動する．

Validではメッセージキューをストリームとしてアクセスすることができ，Valid自体もリスト処理言語であるため，このような処理を簡潔に記述できる．さらにValidでは任意のデータ型をメッセージの要素とできるため，例えばストリーム自身をメッセージとして送ることにより，ユーザ定義のトランザクション処理を簡単に実現できる．

```
module RW_manager()
={for (que,rcnt,wcnt):(nil,0,0) do
  case
    q=[['read|?file]|?rest] →
      if wc=0 then
        { recur(rest,rc+1,wc)
          where !=read(file) }
      else
        recur(q,rc,wc),
    q=[['write|?file]|?rest] →
      if wc=0 and rc=0 then
        { recur(rest,rc,wc+1)
          where !=write(file) }
      else
        recur(q,rc,wc)
  end,
where
  mess=head(message_queue()),
  (q,rc,wc)=
  case
    mess=['read|_] or mess=['write|_]
      → ([que|mess], rcnt, wcnt),
    mess=['r_done|_]
      → (rque, wque, rcnt-1, wcnt),
    mess=['w_done|_]
      → (rque, wque, rcnt-1, wcnt-1)
  end },
```

図4. 読みだし書き込み問題のプログラム

4. まとめ

データ駆動に基づいたメッセージフローシステムによる，プログラミングの例を示した．状態を導入したことにより，純粋な関数性からは後退したが，入出力制御などのデータフローが元来苦手としていた部分の管理を素直に表現できる様になった．例えば動的に増加するユーザからの処理要求(この様な要求はリンクが固定的である従来のデータフローでは扱うことができない)を処理できるResource Manager等を構築することが可能となる．

なお本稿で述べた言語Validは，現在我々の研究室で開発中の超並列マシンDatarol上での主要言語として処理系を開発中である．

5. 参考文献

[1] R.E.Filman and D.P.Friedman:“Coodinated Computing”, MIT Press (1984).

[2] G.Agha:“Actors: A Model of Concurrent Computation in Distributed Systems”,MIT Press (1986).

[3] H.Abelson and G.J.Sussman:“Structure and interpretation of computer programs”,MIT Press (1985).

[4] 雨宮,長谷川:“並列協調システムにおけるメッセージ処理機構”,ソフトウェア科学会第4回大会論文集, pp.315-318 (1987).

[5] 日下部,友清,谷口,雨宮:“関数型言語Validの並列オブジェクト指向プログラミング言語仕様”,信学技報 CPSY90-12-37 (1990).