3Q-5

時相論理に基く並行処理記述用論理型言語の提案

相田　仁，田中　英彦，元岡　達
(東京大学　工学部)

1．はじめに

論理型言語において、並行に処理されるプロセス（ゴール）の間の同期関係を記述する方法としては、大別してデータ依存関係に基づく方式と時相論理に基づく方式とがある。従来、データフローをはじめとして多くの並列処理モデルにおいて、データ依存関係により同期をとることを試みてきたが、各種の同期関係をすべてデータ依存関係によりあらわそうとすると、同期のために導入された本来の処理とは直接関係のないデータが多数のプロセス間を行き交うことになり、プログラムの理解しやすさ、実行の効率などの点において好ましくない結果をもたらす。Interval Temporal Logic をそのままプログラミング言語として用いようとする Tempura[1]の試みは上記の点において大変興味深いが、Tempura はもともと論理回路の仕様を記述し設計がそれを満たしているかどうかを検証するために用いる言語[2]に基づいているので、プログラム言語として用いるにはやや不便な点がある。

また従来 Prolog では assert,retract が各所で用いられてきたが、並列処理の環境下では assert, retract は影響の及ぶ範囲が明確でないため、そのままでは実現することができない。従来の Prolog における assert, retract の用いられかたとしては、よく用いられている順に次のようなものがある。

(1)引数で持ち運ぶのが面倒な大域的な変数の値を格納する。
(2)無駄な重複計算を避けるため一旦行なった計算結果を格納しておく。
(3)プログラムを動的に変更する。

これらのうち、(3)が最も本質的であるが、その実例は少ない。(2)は知的後戻りのような別のメカニズムとして実現すればよく、本来、ユーザプログラムとは切り離して取り扱うべき問題である。そこで、(1)のような場合に対して、プログラミングが容易となるような機能を提供するのがよい。

以上のような点から、ここでは時間とともに値を変える大域変数を導入した論理型言語を提案する。

2．基本言語機能

i）時刻オペレータ

```
p || q  etc.
```

ここで提案する言語においては、Tempura にならってすべてのゴールはある仮想的な interval において処理されるものと考える。時刻オペレータは現在のintervalの中間に一つの時刻を設定するもので、大域変数の値はこのような時刻を期して更新される。Tempura の場合と異なり最小時刻単位は考えず、大域変数の値は一旦設定されると、次に変更されるまではいくつ時刻を過ぎようとも値を保持しつづける。一般に interval の示す仮想的な時間順序と、処理系が計算機内でプログラムを処理・解決してゆく順序とは同じである必要はない。そこで計算機内部でどの順に処理してゆくべきかの指定（アドバイス）として、次の３通りを考える。

p && q	p の解を求めてからq の解を求める
p || q	解の探索順序は規定しない
p !! q	q の解を求めてからp の解を求める

ii）大域変数の参照

```
? x
```

現在対象としている interval の始まりの時刻における大域変数xの値を参照する。

iii）大域変数の設定

```
x := value
```

現在対象としている interval の終わりの時刻における大域変数xの値をvalue に設定する。

並行に動作する各プロセスは必要に応じて大域変数の値を参照・変更し、大域変数を通じて同期をとる。下記に、２つのプロセスの間でハンドシェークによりデータを受け渡しする場合のプログラム例を示す。データの送り手プロセスと受け手プロセスの間にハンドシェーク用の stb, ack の２つの大域変数と、実際にデータを受け渡しするための port 大域変数とを用意する。送り手プロセスはack が０であるのを確認し、 port にデータを書くと同時にstb を１にする。次いでack が１になるまで待ち、stb を０に戻す。受け手プロセスはstb が１になったら port のデータを読み、ack を１にする。stb が０に戻ったらack を０にする。

```
send(X) :-
        true ||
        ?ack = 0, port := X, stb := 1 ||
        true ||
        ?ack = 1, stb := 0 ||
        true.
```

```
receive(X) :-
        true ||
        ?stb = 1, X = ?port, ack := 1 ||
        true ||
        ?stb = 0, ack := 0 ||
        true.
```

ここで、ゴール

```
?- ack := 0 || send(message), receive(X).
```

が与えられた時には時間軸の正の向きに解を探索するのが自然であるが、

```
?- receive(message), send(_), X = ?ack.
```

のように、時間的に終わりの状態が先にわかっている場合には、時間軸と逆向きに解を探索したほうがよいこともある。

3．排他制御

2つのプロセス間でのハンドシェークのように、大域変数に関してそれをを参照するプロセスと設定するプロセスとが固定されている場合には、それらの前後関係のみを与えれば正しく動作するが、複数のプロセスが1つの大域変数に対して参照・設定を行なうような場合には前後関係を定めるだけでは不十分である。例えば前節のプログラムにおいて、複数の送り手プロセスが1つの受け手プロセスにデータを送ろうとする場合には、単に大域変数を共有したのでは、大域変数の参照・設定のタイミングによっては正しく動作しない。

この問題を解決するためには、何らかの排他制御のためのメカニズムをとり入れる必要がある。最少限の機能によりこれを実現する方式としては test and set による方法があり、また Concurrent Prolog[3]で行なわれているように、データへのアクセス要求を merge/distribute するプロセスを間に入れて、直接には大域変数へのアクセスを認めない方法なども考えられる。

ここでは、大域変数への書き込みを排他制御する機能を追加することにより、この問題を解決する。ゴール

```
x ! Goal
```

は、（xが他のゴールによりすでにロックされていないときに）大域変数xを排他ロックし、xへの値の設定を Goal のみに認めつつ Goal を実行することを表わすものとする。この機能を取り入れると、前記のハンドシェークのプログラムにおいて port への同時書き込みを禁止することにより、複数のプロセスからのデータを正しく受け渡しできるようになる。

4．副作用

入出力のような、外界に対する副作用に関しても、大域変数と結びつけて考えることにより取り扱うことができる。例えば次に示す例において、大域変数 screen が端末の画面を表わしているものと考えることにより、端末への出力操作を表現することができる。

```
output(X) :- screen !
        (append(?screen, [X], S), screen := S).
```

外界に対する副作用を実現するためには、処理系内部で得られた仮想的な大域変数の変化の過程を入出力プロセッサなどに送って副作用として順に実行すればよい。その際、大域変数の変化の過程の選択肢が複数あったとしても、その現実的意味はつけにくい。そこで、このような外界への副作用の顕現は、選択肢がただ1つに絞られた場合に限り行なうことにする。そこで、副作用を伴うプログラムは一般的に次のような3つのフェーズで実行されることになる。

i）論理的な解を求めるフェーズ
ii）解を1つに絞るフェーズ
iii）副作用を外界に示すフェーズ

第2のフェーズは、通常、ユーザがプログラムで示したガードにより実現されるが、たまたま解を探索しているうちに選択肢が1つに絞られたならば、そこまでに確定した大域変数の変化を副作用として外界に示してしまってかまわない。

5．おわりに

上記の言語に関しては簡単なインタプリタを作成し、排他制御などが正しく動作することを確認した。効率良い処理系の作成方法などについては今後の課題である。

＜参考文献＞

[1] Moszkowski,B. and Manna,Z.: Temporal Logic as a Programming Language, Proc. of Parallel Computing '83, West Berlin, Sept. 1983.

[2] Moszkowski,B.: A Temporal Logic for Multi-Level Reasoning about Hardware, Report No.STAN-CS-82-952, Dept.of Computer Science, Stanford University, Dec. 1982.

[3] Shapiro,E.Y.: A Subset of Concurrent Prolog and Its Interpreter, Technical Report TR-003, ICOT, Feb. 1983.