ソフトウェア工学 22－12
（1982. 2. 10）

並列型プロセスの仕様を記述するための言語とその実現

瀬川 清（平組田大学），坂東 浩之，野田 晴義，
土居 範久（慶應義塾大学）

1．はじめに

仕様記述のための言語及び方法論は多数提案されている。PDR ── Process Data Representation [5,7] は並列処理の仕様記述の一手法である。PDRでは，並列処理を二つの面 ── プロセス（能動的な面）及びデータ（受動的な面）から捕え，その仕様を記述する。また，際どい演算（critical operation）を記述するため，強制論理（forcing logic）[6] を用いる。

PDRを用いて記述した仕様を計算機により処理しようとすると，そのための言語（一般にPDR言語という）が必要になる。すなわち，PDRによる仕様をPDR言語により，計算機で処理できる形 ── 算譜 ── にする。PDRの概念に沿ったものならば，PDR言語は一つに固定する必要はない。むしろ使用環境に応じたPDR言語が多数あるべきだとさえ言える。

今回PDR言語（の一つ）をDEC20上で実現した。以下，仕様記述法PDR，強制論理及び実現したPDR言語について述べる。

2．仕様記述法PDR

個々に実行される単位をプロセス，相互に強く関連したプロセスの集まりをクラスタと呼ぶ。一つの並列処理はクラスタの集まりからなる。

PDRではクラスタは次のように記述される：

TC [[PC (prelude, interlude, postlude | data)]]

TC：このクラスタに対する制約の記述。クラスタ総合条件と呼ぶ。

PC：このクラスタのプロセスに対する制約の記述。プロセス総合条件と呼ぶ。

prelude, interlude, postlude：PDRではプロセスは3種類ある。このクラスタのプロセスとその起動条件の記述。

data：このクラスタの共用データの記述。

クラスタ総合条件はプロセス側及びデータ側から記述した条件で，強制論理により記述する。一方，プロセス総合条件はプロセス側からのみ記述した条件である。

PDRでは，このようにプロセスを中心とする記述とデータを中心とする記述を同等に扱う。また，プロセス側からの記述で充分ならば，データ側からの記述は省略してよい。逆に，データ側からの記述で充分ならば，プロセス側からの記述は省略してよい。もちろん省略できる場合であっても，そのまま残しておいてもかまわない。このような冗長性はPDRの一つの特徴である。

3種類のプロセス prelude, interlude, postlude は，それぞれ，いつ起動されるかを表わす起動条件を持つ。

クラスタの実行は3種類のプロセスを順に実行することであり，クラスタの状態はそれに応じて以下の三つの状態をとる。

プレリュード状態

クラスタが起動されると，すべてのプレリュード・プロセスの起動条件が評価される。そして，起動条件が真であるプロセスが起動される。この状態は起動されたプロセスがすべて終了すると終わる。

インタールード状態

各インタールード・プロセスの起動条件が真になるまで繰り返し評価される。起動条件が真になると対応するプロセスが起動される。そして，終了すると，また起動条件が評価されるようになる。この状態は，起動されたすべてのプロセスが終了し，すべてのインタールード・プロセスの起動条件が偽のときに終わる。

ポストルード状態

各ポストルード・プロセスの起動条件が真になるまで繰り返し評価される。起動条件が真になると対応するプロセスが起動される。この状態はすべてのポストルード・プロセスが終了すると終わる。そしてクラスタの実行が終了する。

プロセスは，その種類により起動される回数が異なる：

プレリュード・プロセス	0又は1回
インタールード・プロセス	任意回
ポストルード・プロセス	1回

クラスタ間には順序関係のみを規定する。共用データの有無については特に定めない。仕様記述法PDRでは，このように「定めない」という点がある（もちろん後で述べるPDR言語では，きちんと定めてある）。PDRでは仕様記述の中核だけを定めており，他は定めない。これも一つの特徴である。

3．強制論理

強制論理（forcing logic）は演算の性質を，行なうもの ── プロセス ── と受けるもの ── データ ── の数により記述する。強制論理の形式的な定義は[6,7]にあるので，ここでは簡単に定義する。

強制演算子（forcing operator）

次の二つの強制演算子を用いる：

(1) $[x_1, \cdots, x_m]:k$　$\{x_1, \cdots, x_m\}$の部分集合で，要素の数（濃度）がk以下のもの全体．

(2) $\langle x_1, \cdots, x_m \rangle:k$　$\{x_1, \cdots, x_m\}$の部分集合で，要素の数がk以上のもの全体。

例えば，$[x,y,z]:2$ は

$$\{\phi, \{x\}, \{y\}, \{z\}, \{x,y\}, \{y,z\}, \{z,x\}\}$$

を表わす。

二つの演算子を特に区別する必要がない場合は記号 ≪　≫ を用いる。

強制演算子は入れ子にして用いることもできる。例えば，次のようになる：

$$[[x,y]:2, z]:1 = \{\phi, [x,y]:2, \{z\}\} = \{\phi, \{\phi, \{x\}, \{y\}, \{x,y\}\}, \{z\}\}$$
$$= \{\phi, \{x\}, \{y\}, \{x,y\}, \{z\}\}$$

強制式（forcing expression）

強制式は次のようなものである：

(3) $\ll A \gg \xrightarrow{@P} \ll B \gg$

これは$\ll A \gg$の要素の一つ（$\ll A \gg$は中集合の部分集合である）が，演算@Pを$\ll B \gg$の要素の一つに行なうことを表わす。

(3)の$\ll A \gg$として$[x_1, \cdots, x_m]:k$を用いた場合，すなわち強制式

(4) $[x_1, \cdots, x_m]:k \xrightarrow{@P} \ll B \gg$

を考える。強制演算子の定義(1)より，(4)の左辺は

$$\{\phi, \{x_1\}, \cdots, \{x_m\}, \{x_1, x_2\}, \cdots, \{x_{i_1}, \cdots, x_{i_k}\}\}$$

となる。強制式の定義から，この要素の一つが演算@Pを行なう。以上から，(4)は$\{x_1, \cdots x_m\}$のうち「高々k個」が演算@Pを$\ll B \gg$の要素の一つに行なうことを表わす。

同様に，強制式の左辺に$\langle x_1, \cdots, x_m \rangle:k$を用いた強制式

(5) $\langle x_1, \cdots, x_m \rangle:k \xrightarrow{@P} \ll B \gg$

は，$\{x_1, \cdots, x_m\}$のうち「少なくともk個」が演算@Pを$\ll B \gg$の要素の一つに行なうことを表わす。

このことから，強制演算子[ ]を「高々演算子（at most operator）」，$\langle\ \rangle$を「少なくとも演算子（at least operator）」とも呼ぶ。また，強制演算子を括弧演算子（bracket operator）と呼ぶこともある。

強制式の右辺に強制演算子を用いる場合も同様に定義できる。

強制式の左辺に高々演算子を用いた場合と少なくとも演算子で規制子（添字kのこと）の値が0の場合は，$\{\phi\}$が演算を行なう場合も含んでいる。これは意味がないので，以後，考えないことにする（$\{\phi\}$が演算を行なうということは，演算が行なわれていないことであると考えてもよい）。しかし，強制式の右辺の場合は必ずしも意味がないとは言えない（実際には，ほとんどの場合，意味がないと言える）。

さらに，規制子が特別な値を持つ強制演算子を含む強制式を考えると次のようになる：

$[x_1, \cdots, x_m]:1 \xrightarrow{@P}$ $x_1, \cdots, x_m$のうちの一つだけが演算@Pを行なう。これは相互排除（mutual exclusion）を表わす。

$[x_1, \cdots, x_m]:m \xrightarrow{@P}$ $x_1, \cdots, x_m$のうちの高々m個が演算@Pを行なう。これは演算@Pを行なうことを禁じていないことを表わす。

$\langle x_1, \cdots, x_m \rangle:m \xrightarrow{@P}$ $x_1, \cdots, x_m$が演算@Pを行なう。これは協同（co-operation）を表わす。

強制式を用いた同期問題の記述例を示す。これはPDRによる簡単な仕様記述の例でもある。

例．1　読み書き問題（Readers'－Writer's Problem）．

R_1，R_2を二つの読み手，Wを書き手，FILEを共用資源とする。この問題は次のように記述できる。

$$[[R_1, R_2]:2, W]:1 \xrightarrow{USE} \langle FILE \rangle:1$$

例．2　喫煙者問題（Cigarette－Smokers' Problem）．

S_1をマッチを持っている喫煙者，S_2を紙を持っている喫煙者，S_3をタバコを持っている喫煙者，Mをマッチ，Pを紙，Tをタバコとする。この問題は次のように記述できる。

$$[S_1, S_2, S_3]:1 \xrightarrow{SMOKE} [\langle P, T \rangle:2, \langle T, M \rangle:2, \langle M, P \rangle:2]:1$$
$$[S_1]:1 \xrightarrow{SMOKE} \langle P, T \rangle:2$$
$$[S_2]:1 \xrightarrow{SMOKE} \langle T, M \rangle:2$$
$$[S_3]:1 \xrightarrow{SMOKE} \langle M, P \rangle:2$$
$$[S_1, S_2]:1 \xrightarrow{SMOKE} \langle T \rangle:1$$
$$[S_2, S_3]:1 \xrightarrow{SMOKE} \langle M \rangle:1$$
$$[S_3, S_1]:1 \xrightarrow{SMOKE} \langle P \rangle:1$$

各強制式が論理積（and）で結ばれているので，最初の強制式はなくてもよい。

強制論理を同期基本命令とみたとき，従来のものと異なり，作用者／被作用者の数という高級な概念を用いている。PDRによる仕様記述は，このような高級概念を用いた，冗長度（二面記述）を含むものである。このことから，仕様の記述及び検証が容易になることが期待される。

4．PDR言語

PDR言語（仕様記述言語であるとも言える）を設計するときには次のことを考慮しなければならない。

1．中核を生かす。

2．冗長性をある程度，殺すことは勧めるべきではないとしても，冗長性がPDRの大きな特徴であることを忘れてはならない。

これらのことを考慮すれば，「見てくれ」とは関係がなくなる。すなわち，PDR言語と呼ばれるものが多数あってもよい。その結果，PDRによる仕様（これは一つ）から，PDR言語ごとに同じ機能を持つ算譜ができることになる。このことは，ソフトウェアの移植性（Portability）を高めることにもなる。

今回，実現したPDR言語（以後，単にPDR言語と呼ぶ）はSIMPL[1]を基礎として設計したものである。すなわち，SIMPLを拡張した言語になっている。

図.4－1にPDR言語による算譜の一般形を示す。

＊cluster path expression ―― クラスタ順路式

クラスタの実行順序を順路式[3]の記法により表わす。ここでは順路式という術語を用いたが，正確にはプロセス式[6]である。

＊cluster total condition ―― クラスタ統合条件

強制式により，クラスタ内のプロセスと共用データの関係を表わす。

* declation for shared data
—— 共用データ宣言

クラスタ内の共用データを宣言する。同時に強制式により，どのプロセスから参照されるかということも表わす。

* process total condition
—— プロセス総合条件

順路式記法により，プロセスの実行順序を表わす。この条件はプロセスの起動条件の一部とみなす。

* activation condition ——
起動条件

論理式により，プロセスが実行される時点を表わす。

* data requirement —— データ要求

強制式により，プロセス内で参照する共用データを表わす。

3箇所で強制式を使用しているが，すべて記述する必要はない。

隙どい演算は，"演算名::(実行文の並び)" と表わし，強制式"A $\xrightarrow{\alpha P}$ B"は"αP(A,B)"と表わす。

クラスタ間に共用データはない。必要ならば外部ファイルを用いる。このファイルに対する参照は使用者の責任で行なうことになる。

PDR言語による，「五人の哲学者の食事問題」の算譜を図.4-2に示す。

```
program  program_name
   cluster path expression /* specify the relation
                              among clusters using
                              path expression notation */

/* specification of a cluster */
cluster  cluster_name
   cluster total condition  /* using forcing logic */

   /** data specification **/
      declaration for shared data /* using forcing logic */

   /** process specification **/
      process total condition /* specify the relation
                                 among processes using
                                 path expression notation */

      /* process declaration */
         prelude process  process_name
           (or interlude process or postlude process)
            activation condition /* Boolean expression */

            data requirement /* reference declaration
                                for shared data using
                                forcing logic */

                  program body  /* a complete SIMPL program
                                   corresponding to
                                   this process */
```

図.4-1　PDR算譜の一般形

5. PDR言語の実現

実行時支援系として，PDR算譜全体を管理するものとクラスタごとにクラスタを管理するものを用意する。これらが，条件の評価を行なう。共用領域はクラスタ管理算譜中に確保し，クラスタ内のプロセスはTαPS-20 [4]のPMAPシステム呼出しにより，その領域を使用する。

次に典型的な強制式の実現方法を述べる。今回の実現では，強制式の方辺(データ側)は$\langle d_1,\cdots,d_m\rangle:m$の形のものに制限した。この制限の下

```
program  Dining_Philosophers

   cluster Main

      data specification
         INT FORK1 : eat ( [PH5,PH1]:1 , )
         INT FORK2 : eat ( [PH1,PH2]:1 , )
         INT FORK3 : eat ( [PH2,PH3]:1 , )
         INT FORK4 : eat ( [PH3,PH4]:1 , )
         INT FORK5 : eat ( [PH4,PH5]:1 , )
      end data specification

      process specification
         interlude process PH1
            forcing eat ( , <FORK1,FORK2>:2 )
            PROC PH1_MAIN
               INT TIME
               WHILE true DO
                  WRITE ('PH1 THINKING')
                  TIME:=RANDOM
                  WHILE TIME > 0 DO TIME:=TIME-1 END
                  WRITE ('PH1 HUNGRY')
                  TIME:=RANDOM
                  eat :: ( WRITE ('PH1 EATING')
                           WHILE TIME > 0
                              DO TIME:=TIME-1 END )
               END
            START PH1_MAIN
         end process PH1

         /* same specifications for
               processes PH2,...,PH5 */
               .
               .
               .

      end process specification
   end cluster Main
end program  Dining_Philosophers
```

図.4-2　PDR算譜例

では，データを確保することと解放という演算を実行することを同一視できる。よって，強制式の左辺（プロセス側）のみを実現すればよい。

$[x_1, \cdots, x_m] : e \xrightarrow{op}$ は図.5-1のように実現する。ここで，sは強制演算子に対する初期値をのカウンタ，opは演算OPに対する拡張2進セマフォとする。ここで用いているcause/awaitと命名はBrinch Hansenのもの([2])を拡張したものである。

$\langle x_1, \cdots, x_m \rangle : e \xrightarrow{op}$ は逆セマフォ系により実現できる([7])。

強制演算子が入れ子になっている場合は内側から順に展開していけばよい。

```
type extended_binary_semaphore =
   record
      counter :  integer ;
      wait_queue :  queue ;
      event_queue :  queue
   end

prologue
      P(op) ;
         if s > 0 then s := s - 1
         else await(op)
         fi ;
      V(op) ;

code for the critical operation OP

epilogue
      P(op) ;
         s := s + 1 ;
         cause(op) ;
      V(op) ;
```

図.5-1 強制式の実現

6. まとめ

PDRは強制論理という高級概念と冗長性による人間向きの仕様記述法である。計算機上でPDR仕様を処理するためのPDR言語を実現した現在，PDRは並列処理の記述系であると言うことができる。

参考文献

[1]. Basili, V.R. and Turner, A.J., "SIMPL-T: A Structured Programming Language", University of Maryland, Computer Science Center, CN14.2, 1975.

[2]. Brinch Hansen, P., "Operating System Principles", Prentice-Hall, 1973.

[3]. Campbell, R.H. and Habermann, A.N., "The Specification of Process Synchonization by Path Expressions", Lecture Notes in Computer Science, Vol. 16, Springer-Verlag, 1974, P. 89-102.

[4]. DEC, "TOPS-20 Monitor Calls Reference Manual", AA-4166D-TM, Digital Equipment Corporation, 1980.

[5]. Hirose, K., Saito, N., Doi, N., Segawa, K. et al., "Process-Data Representation", Proc. of 3rd UJCC, 1978, P. 225-230.

[6]. Hirose, K., Saito, N., Doi, N., Segawa, K. et al., "Forcing Logic in Process-Data Representation", Technical Report KIIS-79-01, Institute of Information Science, Keio University, March 1979.

[7]. Hirose, K., Saito, N., Doi, N., Segawa, K. et al., "Specification technique for parallel processing : Process-Data Representation", Proc. of AFIPS 1981 National Computer Conference, Vol. 50, May 1981, P. 407-413.

[8]. Lauer, P.E. and Campbell, R.H., "Formal Semantics of a Class of High-Level Primitives for Coordinating Concurrent Processes", Acta Informatica 5, 1975, p. 297-332.