ソフトウェア工学 66－4
（1989. 6. 27）

哲学者の食事問題のLOTOSによる記述実験

大蒔和仁(電子技術総合研究所 情報アーキテクチャ部)
五反田隆広、小野昌秀、佐藤嘉一(沖電気工業(株) コンピュータシステム開発本部)
藤田朋生、新田徹(日本電気(株) 基本ソフトウェア開発本部)
田中功一(三菱電機(株) 情報電子研究所)
堀田英一(日本電信電話(株) NTTソフトウェア研究所)
五ノ井敏行(富士通(株) 情報システム事業本部)
内山光一((株)東芝 情報通信システム技術研究所)
島田明宏(シャープ(株) 技術本部)

通信プロトコルOSIのための形式的仕様記述言語LOTOSがISO8807として制定された。我々はLOTOSの妥当性を探ることを目的としてLOTOS研究会を行なっている。そこではLOTOSの記述のためのGuidelineの調査、LOTOSによる実際のプロトコルの記述実験、LOTOSに関係する文献調査、等を行なっている。

研究会の構成メンバは、(1)実際にOSIのプロトコル開発に携わっている人、(2)プロセス記述の理論的な研究をしている人、(3)LOTOS以外の形式記述言語に詳しい人、(4)C等の言語プロセッサ開発に携わっている人、(5)LOTOSが提案され出したころから勉強している人、等から成っている。

研究会ではLOTOSとはどんな言語であるかを知るための練習問題として「哲学者の食事問題」を取り上げ、幾つかの記述実験を行なった。本稿はこの記述実験をとおして我々が得たLOTOSの仕様記述言語としての感想を述べる。LOTOSによる「哲学者の食事問題」の解法についてはすでに文献に現れているが、本稿の目的は初めてLOTOSに触れた人がこの有名な問題に対してどのように取り組んだかを示し、LOTOSに対して抱いた率直な感想や意見を述べることにある。

Experimetal Descriptions of Dining Philosopher's Problem in LOTOS and Its Experience

K.Ohmaki (Electrotechnical Laboratory)
T.Gotanda, M.Ono, and Y.Sato (Oki Electronic Industry Co.,Ltd.)
T.Fujita and T.Nitta (NEC Corporation)
K.Tanaka (Mitsubishi Electric Corporation)
E.Horita (Nippon Telegraph and Telephone Corporation)
T.Gonoi (Fujitsu Limited)
M.Uchiyama (Toshiba Corporation)
A.Shimada (Sharp Corporation)

LOTOS is a language to specify OSI protocols formally. LOTOS has become ISO 8807 this February. We are organizing a research group for the language LOTOS, and studying its feasibility. Up to now, we are surveying the Guideline for LOTOS specification, writing a real OSI protocol in LOTOS, and reading papers related to LOTOS.

The members of the group are (1)actually developing OSI protocols, (2)doing theoretical works on process description, (3)familiar with other formal description languages, (4)developing language processors like C compiler, or (5)studying the language specification of LOTOS at the early stages.

To know what LOTOS is, we have described the dining philosphers problem by several approaches. In this paper, we present our impressions on LOTOS through these experiences. Although the dining philosoper's problem has already been written in literature, the main purpose of this paper is to show how we approach the specifications for the first time as LOTOS users. We also give the controversial comments on LOTOS.

1 まえがき

通信プロトコルOSIのための形式的仕様記述言語LOTOSがISO8807として制定された [1]。LOTOSサブセットのシミュレータはすでにオランダ等において作成されている [2]。国内でもLOTOSの検証系等の処理系に関する研究が行なわれている [3,4,5]。また、LOTOSを通信プロトコル以外のより一般的な問題の仕様記述にも用いようとする研究も行なわれている [6]。

我々はLOTOSの妥当性を探ることおよびLOTOSの処理系としてはどんな要求仕様が考えられるかを調べることを目的としてLOTOS研究会を行なっている。研究会の構成メンバは、(1)実際にOSIのプロトコル開発に携わっている人、(2)プロセス記述の理論的な研究をしている人、(3)LOTOS以外の形式記述言語に詳しい人、(4)C等の言語プロセッサ開発に携わっている人、(5)LOTOSが提案され出したころから勉強している人、等である。

研究会ではLOTOSとはどんな言語であるかを知るための練習問題として「哲学者の食事問題」を取り上げ、幾つかの記述実験を行なった。

本稿は、我々の研究会の様子を紹介し、この記述実験をとおして我々が得たLOTOSの仕様記述言語としての感想を述べる。LOTOSによる「哲学者の食事問題」の解法についてはすでに文献に現れているが [9]、本稿の目的は初めてLOTOSに触れた人がこの有名な問題に対してどのように取り組んだかを示し、LOTOSに対して抱いた率直な感想や意見を述べることにある。LOTOSの構文や意味については [1]を参照されたい。

2 研究会で行なっている内容

LOTOS研究会では本年1月初めより、ほぼ2週間に一度の割合で開催している。この研究会では次のような活動を行なっている。

1. LOTOS記述のガイドライン読み合わせ: 文献 [7]には4種類の例題についてEstelle, LOTOS, SDLの3種類のFDTを用いて記述が比較されている。このうち、LOTOSに関する部分について詳細な読み合わせを行なった。その結果LOTOS記述で用いられる基本的な手法は理解した。また、小規模のプロトコルのLOTOSによる記述方法についても理解した。これらの記述例には比較的多くのバグや問題点があり、それらについて検討し記録に残した。

2. 簡単な例題書き: LOTOSで実際の問題の記述を行なってみて感覚を掴むことを目的として、本稿で述べる哲学者の食事問題を取り上げ、何人かで独立に記述してみて、LOTOSの記述経験を深めた。この記述はHIPPOと呼ばれるシミュレータ [2]を用いて動作実験を行なった。

3. 実際のプロトコルの記述: 本物のプロトコルをLOTOSで記述してみるべきだと考え、ACSEを例題にしてLOTOSによる記述を行なっている。これは初期バージョンはほとんど完成しており、今後この記述を研究会で叩いて、LOTOSのプログラミングスタイルについて検討を加える予定である。

4. 文献調査:

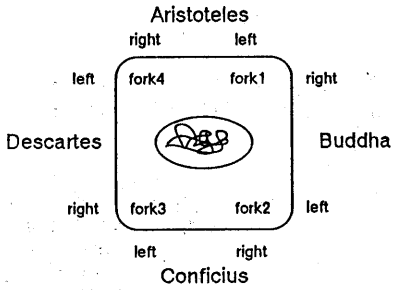

Figure 1: 哲学者の食事問題

特に [8] を中心にLOTOSに関する文献調査を行なっている。今までに、実際にLOTOSシミュレータを構築した経験、LOTOSをソフトウェア開発に実際に適用した例、LOTOSからテストスーツを生成する方法、などを調査した。

3 記述の試み

ここでは、図1に示すように、4人の哲学者が4本のフォークを使い食事をするものとする。この問題設定は文献 [2] よりとった。LOTOSによる記述練習としてこの問題を取り上げ、次の3種類のアプローチを行なった。

3.1 記述例 (その1)

3.1.1 記述の方針

この記述では、哲学者の食事の問題をまず最初にCCSで記述し、次にそれをLOTOSに翻訳するという方針で行なった。これを通してCCSとLOTOSの記述能力の比較、及び両者の対応関係を考察した。

CCSによる記述は原理的には実行可能 ([11] 参照) であるのに対し、LOTOSは制限指向仕様記述言語の性格を持ち、必ずしも実行可能ではない (4.1節参照)。

3.1.2 記述の概略

食事する哲学者の問題には幾つかの解法が提案されている。問題は相互排他的に使用される資源、数本のフォーク、を用いてdeadlockすることなく如何に数人の哲学者が食事と思索を続けられるようにするかということである。

よく知られている解法としては、相互排除のためにmonitorを使用するもの (例えば [13]) 、資源であるフォークをプロセスとみなすことにより相互排除を実現するもの (例えば[14]) 等がある。

ここでの解法は、上記の2つの方法と異なり、哲学者を表すプロセスが資源 (フォーク) の使用状況を常に認識すること、及び各哲学者に対して第一に取るべきフォークと第二に取るべきフォークを適当に割り振ることにより相互排除とデッドロックの回避を実現するものである。これを提案するに当たっては、本稿の共著者である小野の資料 (当LOTOS研究会資料、1989.2.22. 沖電気) に示唆を得た。

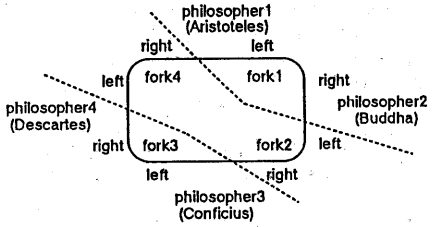

Figure 2: デッドロック回避の手立て

3.1.3 システム構成要素

プロセスとしては、4人の哲学者を表す
philosopher1, ... , philosopher4
がある。これらは
philosopher[Thinking,Eating,Fork1,Fork2]
(name,available1,available2)
というプロセスのひな型から適当な引き数を設定することによってインスタンシエイトされる。ここで、[Thinking,Eating,Fork1,Fork2]はゲートとしての引き数を表し、(name,available1,available2)は値の引き数を表す。ここで、Fork1は第一に取るべきフォークであり、Fork2は第二に取るべきフォークである。また、available1はFork1が使用可能か否かの認識を表す引き数である。

3.1.4 相互排除とデッドロック回避のプロトコル

4人の哲学者
philosopher1, ... , philosopher4
と4本のフォーク fork1, ...,fork4 が forkN の右側に forkN がくるように丸テーブルに配置されているとする (N = 1,.., 4)。(図2参照)

ここで奇数番目の哲学者は右のフォークを第一にとり、偶数番目の哲学者は左のフォークを第一に取ることによって、全ての哲学者が第一のフォークを持った状態で第二のフォークを待つことによるデッドロックを回避することができる。

3.1.5 哲学者の動作

哲学者は available1, available2という2つの内部変数を持つ。これらは共に Boolean の型を持ち、初期値はともに true である。意味は前節で述べた通りである。

哲学者は 次の4つのモード
thinking-mode, eating-mode,
waiting-mode, putting-mode
を持ち、これらのモードをこの順序で繰り返し経過する。いずれのモードにおいてもこの2つの変数available1, available2の値を参照して動作を決定する。

いずれのモードにおいてもゲートFork1, Fork2におけるイベントはゲートを挟んだ隣の哲学者プロセスと同期して行われる。

3.2 記述例 (その2)

3.2.1 全体の構成図

図3に全体の構成図を示す。矩形で囲んだものがプロセスであり、各線に付随しているpfab等の記号

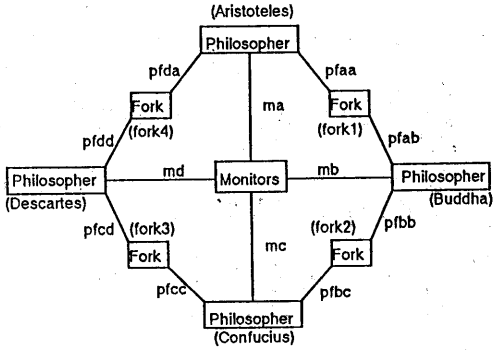

Figure 3: プロセス関連図

が各プロセスの交信を行なうためのゲートである。図3の他にも細かなプロセスがある。

3.2.2 プロセス

各プロセスの動作を説明する。

1. Philosopher:哲学者1人をあらわすプロセス。4人の哲学者の動きは全く同様なので、一般的な哲学者のプロセスを記述しておいて、ゲートにより4人の哲学者を区別する。個々のプロセスPhilosopherは左右のフォークをあらわすプロセスRLFork (図には書いていないが) と、ゲートpfxxで同期している。また、ゲートmxでMonitorsと同期する。

2. Fork:フォークの状態をあらわすプロセス。どのフォークも動作は同じなので、一般的に定義しておいて、ゲートで識別する。フォークの状態は論理型であらわし、true ならばフォークが使われていることをあらわし、false ならば使われていないことをあらわす。

3. Monitors:デッドロックが起きないように、フォークを取りにいける哲学者の数を制限する見張りをしているプロセス。各々の哲学者と同期している4つのサブプロセス Monitor からなる。Monitor の動作は共通なので、一般的に定義しておいてゲートで各々の哲学者に対応するプロセスを識別する。

3.3 記述例 (その3)

3.3.1 外部観測事象

哲学者の動作を外部から観測したいと考えたので、4人の哲学者をゲートとして用意した。そして、哲学者の状態を示す定数値 (思考中/食べたい/食事中) が値としてつく。

3.3.2 プログラムの考え方

デッドロックを回避するには、食事をする哲学者の数を制限すれば良い、ということは知っていた。そこで、違う方法を考えようかと試みた。当方が受け取った出題では、フォークの使い方に何の制限も課せられていなかったので、デッドロックになった

```
specification Dining_Philosophers
  [Aristoteles,Buddha,Confuclus,Descartes] : noexit
...
behavior
  hide
    fork1,fork2,fork3,fork4 in
    ( philosopher[Aristoteles,fork4,fork1]( a ) |||    (* 個々の哲学者の動作 *)
      philosopher[Buddha     ,fork1,fork2]( b ) |||    (* を記述           *)
      philosopher[Confuclus  ,fork2,fork3]( c ) |||
      philosopher[Descartes  ,fork3,fork4]( d )
    )
    |[fork1,fork2,fork3,fork4]|
    ( forks[ fork1 ]( not_used , none ) |||            (* 全体の制約を記述  *)
      forks[ fork2 ]( not_used , none ) |||
      forks[ fork3 ]( not_used , none ) |||
      forks[ fork4 ]( not_used , none )
    )
where
(* 哲学者一人の動作  :  考えてから食べる動作の繰り返し *)
  process philosopher[name,left_hand,right_hand]( n : id ) : noexit :=
    thinking    [name,left_hand,right_hand]( n ) >>
    eating      [name,left_hand,right_hand]( n ) >>
    philosopher[name,left_hand,right_hand]( n )
  where
    ...
  endproc
(* フォークの状態の記憶  :  状態のセットと問い合わせの二処理ある   *)
(*   セットする時は，使用者を確認する．他の人から確保に来た時は    *)
(*   そのフォークへの要求を失敗させる．                          *)
  process forks[fork]( status : use , user_id : id ) : noexit :=
    ...
  endproc
endspec
```

Figure 4: LOTOSで書いたプログラム

と判断した時にすでに手に持っているフォークを置き直し、食事を一時的に諦めるようにすればデッドロックを回避できると考えた。デッドロックと判断するのは、一つのフォークを二人が使用しにいった時、すなわち、手がぶつかった時である、とした。

3.3.3 プログラムの解説（図4参照）

プログラムは大きく、データ定義部とプロセス定義部から構成される。データ定義部は、哲学者の状態（思考中／食べたい／食事中）とフォークの状態（使用中／未使用）と利用者IDと通信用コマンドを定義している。

プロセス定義部は大きく、哲学者の動きを記述した部分と、フォークの状態を記憶する部分に分かれる。両者は、フォークのゲートを通じて同期している。

哲学者の動きを表すプロセスは、思考中と食事中を交互に繰り返す。ただし、フォークが使用できないと、食べたい状態で待つ。

フォークの状態を表すプロセスは、フォークの状態（使用中／未使用）と使用中の時は利用者も記憶している。哲学者プロセスからフォークの状態を問い合わされると、記憶しているフォークの状態を通知する。状態の設定時に、複数の哲学者がフォークを使用しにいったらデッドロック回避シグナルにより食事を中断させてしまう。

4 記述実験の感想

ここでは、上記の記述実験および当研究会で行なってきた文献調査等で抱いたLOTOSに対する率直な感想を述べる。

4.1 Aさんの感想（CCSと比較して）

4.1.1 同期通信のモデル化における違い

LOTOSの起源は、プロセス制御・通信を表すためのCCSとデータを記述するACT1の組合せである。しかし、LOTOSのプロセス制御・通信部分とCCSは同期通信という点では一致しているが、同じではなく、一番基本的な違いとしては次の点がある。

CCSでは2つのプロセスPとQがゲートAにおけるイベントで同期し、そこでのイベントは外からは見えないということをあらわすには、

$$(P|Q)/A \ldots (1)$$

と記述する。ここで、(P|Q)はPとQの並行合成を表し、/Aはイベント A を取り除く操作(restriction)を表す。

一方、LOTOSにおいては同じ動作を

$$\text{hide A in } (P|[A]|Q) \ldots (2)$$

と記述する。この点に注意すればCCSによる記述をLOTOSの記述に逐語的に翻訳することが出来る。

ここで、注意すべきは逆の翻訳は一般には出来ないということである。例えば(1)と(2)は同じ動作を表すが

$$P|Q \ldots (1')$$

と

$$P|[A]|B \ldots (2')$$

はまったく異なる動作を表す。違いの一つは、(2')ではゲートAにおける同期イベントが外から見えるが(1')では見えない。従って(2')ではゲートAで他のプロセスと更に同期することが出来るが、(1')では出来ない。従ってLOTOSでは3者以上のプロセスの同期が自然に記述できるが、この記述を直接的CCSへ翻訳することは出来ない。

4.1.2 実行可能性についての違い

LOTOSの実行が困難である原因として、データ型の実現が困難であることに帰せられる場合が多いが、プロセス制御の部分のみについても実行を困難にする要素が多い。例えばLOTOSにおいてはchoice構文があるが、これを用いて例えば次の様な記述ができる。

$$\text{choice } x_1,\ldots,x_n\text{:int } [] \ [E(x_1,\ldots,x_n)=0] \rightarrow \text{g; stop} \ldots (3)$$

ここでnは任意の自然数、$E(x_1,\ldots,x_n)$は変数$x_1,\ldots,x_n$を含む整数係数多項式である。このプロセスがイベントgを起こすのと方程式

$$E(x_1,\ldots,x_n)=0$$

が解を持つことは同値であるが、これは判定できないことが知られている（Hilbert第10問題の非可解性）[12]。

CCSが実行可能である[11]が、LOTOSはプロセス制御部分に限っても一般には実行可能とは限らないが、これは上記のchoice構文を等を導入して記述能力を上げたためである。LOTOSのこのような豊富な記述能力によって、制限指向仕様記述を容易になるが、一方LOTOS仕様を基にしたシミレーションやテストを行おうとするときは上記のような問題がある。目的によって言語のサブセットを定め、その範囲のみを用いるようにする必要があるかもしれない。

4.2 Bさんの感想

Dining Philosopherの記述を通して、初めてLOTOSを記述してみた。少し前からLOTOSには興味を持っていたが、実際に記述してみてとまどったり、誤解をしていたのが分かったり、いろいろな失敗をした。

LOTOSはとにかく見なれない言語である。どうしてもEstelleと比較してしまうからかも知れない。EstelleはPascalに似ているため比較的入っていきやすい。（実際にはそうでもないのだろうが。）今ま

で幾つかのプログラミング言語での記述経験がないでもない私だが、そうした経験はLOTOSの記述にはあまり参考にはならないかも知れないと思う。やたらに先入観がない方がよいかも知れない。

まず、ゲートの概念が理解しにくい。最初私は、ゲートは各プロセスに付随していてプロセス間で各プロセスのゲートがバインドされるのかと思っていた。これはLOTOS以外の言語で見られる考えであり、LOTOSとは違う。次に、ゲートにおけるデータの流れに方向性があるものと思っていた。つまり、例えば、演算！で出力し、？で入力するものと思っていた。しかしこれも間違いであって、演算子？でも出力できると知ったときも驚きであった。さらに、ゲートにはバッファ機能がないと言うのも実際に記述してみて初めて知ったことである。

イベント列の記述もよく失敗した。選択演算子にしなければならないのに、並列演算子にしてしまったり、4人の哲学者のゲートを通しての同期のとりかたが分からなかったりした。もっとも、こちらはLOTOSというよりも、今までに並列プログラムを書いたことがなかったためによるところが大きいかも知れない。しかしながら、一度馴れてしまえば、LOTOSはプロセスを結合する演算子がかなり豊富であるがゆえにプロセス記述に関しては、かなり強力な言語のように思う。

データ型の書き方はかなり難しい。多分、抽象的に記述するからであろう。しかし、そうだからこそ、特にあるデータ型を定義しようとした時にどのような演算子を定義しておけばそのデータ型を自分が期待している型として表現したことになるなるのか見当がつかない。自分ではうまく表現したつもりでも読む人にとっては果たしてその様に読んでもらえるのであろうかあまり自信がない。書き方がいろいろありそうで、どう書こうかとあれこれ悩むことも多い。大体データ型のライブラリが少し貧弱ではないだろうか。もっと多くの型を予め定義しておいてはどうだろうか。せめて普通のコンピュータ言語にある型くらいはライブラリとしてサポートしておきたいものである。さらにOSI応用層の記述を行なおうとするならば、ASN.1内のデータ定義はサポートしておきたい。そうしておかないと記述の度に定義し直さなければならず、無駄な時間を使うことになると思う。

記述するのが楽ではないのみならず、LOTOS記述を読むのも決して楽とは言えないと思う。特にデータ定義のところが辛い。なんとなく意味を読み取りながら解読していくしかないのであるが、果たしてこの読み方で良いのかと不安になることもある。自分で書いたものも少し長くなると結構辛い。

LOTOSがコンピュータ言語である以上、LOTOS記述を助ける処理系は是非必要であろう。最低パーサとシミュレータ。読みづらいと言うこともあり、この2つは欲しい。そうでないと果たして自分が記述したものが正しく動作するのか調べるのも大変である。

LOTOSは、現在いわゆるFDTの一種である。将来ともFDTであり続けるのだろうか。それともLOTOSが基となって、より強力な分散処理用の実装言語へと発展していくのであろうか。現在のLOTOSは決して使いやすい言語とは私には思えないが、その豊富なプロセス記述能力、データ定義の抽象性の高さ、さらにしっかりした数学的な背景ゆえに割合気にいっている。出来ることならLOTOSを核にした、より強力な実装も可能な様な分散処理言語を設計してみたいものだと思う。

以下に、私が現段階のLOTOSで疑問に思っていることを幾つか記述してみよう。理解不足のためのナンセンスな疑問もあるかも知れない。(1)プロセスのスコープルールはあるのか。また、不要か。(2)プロセスのライブラリのようなものは不要か。また、作れるか。(3)OSIドキュメントの状態遷移表をイベント列として記述することに限界はないか。(4)(3)とも関係するが、状態遷移表とそれを記述したイベント列との間の関係は見やすいものか。(5)データ定義のスコープルールはあるのか。また、不要か。(6)データで意義のライブラリをどのように作っていったら良いか。(7)データ定義の拡張性はスムーズに行なわれるか。例えば、ASN.1の基本的な型からそれらを使って組み上げた構造型を比較的容易に構成できるか。また、自然数から整数、有理数、実数、複素数への拡張などは比較的容易にできるか。(7)ゲート変数のようなものは不要か。また、考えられるか。(8)ゲートを使った同期機構以外の同期のメカニズムは不要か。(9)プロセス関のデータの受渡しでバッファ機能はいらないか。

いろいろ書いてきたが、今後もLOTOSが素晴らしい言語に発展することを期待する。

4.3 Cさんの感想(Guidelineを読んで)

4.3.1 論理式が分かりにくくパズルのようである

これは二つの理由があると思われる。a)記述する側が下手くそ、b)読み手が馴れていない。これらを回避するには、表現のパターン集を作り、読み手書き手の思考構造をなるべく同じにする努力をすることが必要ではないか。

例えばLOTOSは通信プロトコルを記述するための言語であるのにもかかわらず、通信記述に向いたデータ型を標準的にライブラリとして持っていない。そのために、各々の記述が冗長になりやすく、しかも分かりにくくなっている。

4.3.2 コメントのない部分の理解はかなり手間取った

表現の多くは、直接的表現ではなく間接的表現であるため、意図する内容が分かりにくい。これは、書き手の問題と言うよりも言語の性格上の問題でもあろう。しかし、これは反面長所ともなっているため単純に否定することはできない。LOTOSの場合、他の言語以上にコメントなしでは、記述内容の意味付けが難しいものと思われる。

4.3.3 プロトコルが完全に記述されているか分かりにくい

部分的に言いたそうなことは、大体理解できるが、全体的に正確な記述（文法的な意味ではなく論理的な面から）が出来ているかは、まだ十分確認したとは言えない。このためには、相当な時間がかかると思われる。

4.3.4 LOTOSの行く末

LOTOSを使ってプロトコルの記述は、（多分）できると思われる。それを前提にして、LOTOSを使ってプロトコルを記述した場合、どのような人が恩恵を受け、どのような人が混乱するかを考えてみると、多分次のようになるのではなかろうか。

プロトコルの記述者は、プロトコルの記述が簡単になる（かどうかは、どちらかと言うと好みの問題かも知れない）、特にプロトコルの機能追加に関してはかなりその恩恵に預るであろうと思われる。

しかし、その他大勢であるプロトコル利用者にとっては、あまり有難みはなさそうである。つまり、LOTOSの完全なコンパイラができない限り、他の言語に記述された内容を写像する必要があるからである。さらに、LOTOSの完全なコンパイラの作成が現在の技術ではかなりの無理があると思われるため、諦めざるを得ない現状である。そこでもし、他の言語に写像する場合を考えると、例えばLOTOS記述での再帰的記述は、大部分は他の記述に書き換えなければならない。また、網羅性を高めるために、必要なら（大概必要となる）状態遷移表を作らざるを得ないであろう。さらに、必要なデータをメッセージの形にするための解析も必須となるであろう。

と言うことは、プロトコルに関する大多数の人間である利用者側からすると、LOTOSコンパイラか、LOTOSから状態遷移表を作る等の、他の言語への写像のためのツールができない限り、あまり嬉しくはなさそうである。また、どちらにしても直接LOTOSで記述されたものを（真面目に）読むこともなさそうである。

要するに、もう少し、LOTOS導入の損得を考えた上で、何をなすべきかを考える必要がありそうだ。

4.4 Dさんの感想

4.5 データ型の記法とインプリメンタ

抽象データ型（ADT）の記述を見てインプリンメタがある実装用言語（Cのような）でプログラムすることを考えてみる。

例えば整数を定義するADTの中で次のような等式が出てくる [7]。

$$inc(dec(n)) = n ;$$
$$dec(inc(n)) = n ;$$

このincとdecの定義はそれぞれ「1加えること、1引くこと」と解釈したい。ここで解釈とは「このLOTOS記述を与えられたインプリメンタが具体的にどのようなコードに落とすかを考え、なんらかの具体的な関数を割り当てること」とする。しかしながら「2加えること、2引くこと」という解釈も可能であり、無限の解釈が存在する。

ところで解釈として「0を加えること、0を引くこと」としても、inc, decの等式は満足している。そこでこの解釈に従ってインプリメンタがプログラムを組もうとする場合も考えられる。この解釈ではincでもdecでも常に0しか返らないため、incやdecにより、異なる値を表したいとする使用記述者の意図が反映されなくなってしまう。

この解釈の誤りは暗黙の内に

$$inc(n) = n ;$$
$$dec(n) = n ;$$

を付け加えてしまったことによるものと考えられる。従って、LOTOSの仕様記述行なう人は

$$inc(n) \text{ eq } n = \text{false} ;$$
$$dec(n) \text{ eq } n = \text{false} ;$$

のような規則を追加しておいた方がインプリメンタに対して親切であると考えられる。

4.5.1 このことに対する議論

データタイプ宣言をinitial algebraに基づいて与えるということがIS8807に書かれている。従ってインプリメンタもそのことを十分理解してLOTOS記述を読むべきであろう。

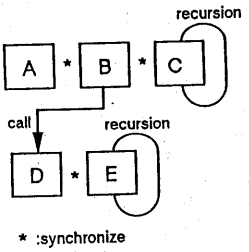

Figure 5: プロセスの絡み合い

LOTOSの記述（特にデータタイプの宣言）が何を意図して書かれているのかを読み取ることが難しいということを、LOTOSの記述を読みはじめた時から感じている。

4.6 Eさんの感想

4.6.1 再帰呼び出しの見づらさ

プロセスの再帰呼び出しを使った記述の場合、特に同期処理が入ってくると、その動きを机上で読み取るのに労力がかかる。例えば図5において、プロセスA,B,Cは各々同期して動くように記述され、Cは再帰的に定義されているとする。また、Bはその中からDを呼び、EはDと同期して動くように記述され再帰的に定義されているものとする。

このとき、例えばEの振舞いを机上で読み取ろうとする時、A,B,C,D,E全部のプロセスの振舞いを知らないと理解できない。

4.6.2 アクションの透明さ

例えば「ConReqを発行する」といったアクションが「!ConReq」と言った具合に簡単に記述されることになり、なんとなく頼りない。アクションはどこに記述されているのかと、探してしまった。

4.7 Fさんの感想

LOTOS言語を使用することにより、楽しくプログラミングすることができたと思う。すなわち、記述しようと思うことを素直に表現できる。使ってみて面白いと感ずる言語である。この哲学者の問題に関しては、LOTOS言語がこの問題をプログラミングするために必要な機能をすべて提供しているため、プログラミングも容易であった。

しかし、LOTOS言語はプログラミングミスをおかしやすい言語仕様を持っているように思えて仕方なかった。これはLOTOSには並列の演算子が沢山あり、それらの間の優先順位が少し複雑であること、また、ADTにより演算子を自由に定義できるため、似たような名前（または同じ名前）を異なる意味で使うことができる、等の性質を持っているからである。

4.8 Gさんの感想

4.8.1 記述

『哲学者の問題』の記述例からもわかるが、LOTOSは今までの仕様記述言語とはちょっと雰囲気の違った言語であり、システムをLOTOSの、どのような表現に対応させてよいのかがはっきりとしない(迷ってしまう)。

慣れるまでの辛抱だと言われたらそれまでであるが。それにしては、使えるようになるまでに本来の目的と違った努力を強いられる言語だなと思った。

inital argebra に従っていると言っても、普通のインプリ屋さんにわかるのだろうか？それとも、LOTOSぐらい理解できない人間はこれから先、インプリする資格などない？

4.8.2 読み易さ

LOTOSによる階層的なシステム記述は確かに美しいと感ずる。しかし、書いてあるものを理解する(LOTOS記述を読むこと。特に第3者が)場合を考えると、結局ボトムアップ的になって、結果として表現に見なれない部分があることもあって、余計、時間が掛かる様な気がする。特にLOTOS記述を行なった人がやたらとLOTOSの癖(?)をうまく使ったような表現ばかりしたら、読む人にとって、どうであろうか。(特にADTの部分は他人に説明して貰わないと何を意味しているのか判らない部分が多い)

4.8.3 インプリメント

トップダウン的にシステムを記述したり、抽象的な表現は理解の助けになるであろう。しかし、実際の処理系を作ろうとすると、LOTOSの表現で美しかった部分が障害となるのは明白である。おおよそ、言語屋さん以外は'動くもの'が無い限り使わないだろうし、できたとしてもよほど効率の良いコードを吐くか、世界的標準にでもならない限りつかわないであろう(時間が解決してくれる?)。

4.8.4 まとめ

LOTOSを導入すると何処がどのように良くなるのかが実感として摑めないというのが率直な意見である。いずれにせよ、新しい言語(?)だけに世の中に(本当に使われる)受け入れられるには、もっと参考になる文献が出ないと無理かな？と思っている。

4.9 Hさんの感想

4.9.1 プロセス定義について

プロセス定義が階層的に記述できるため、あるLOTOS記述を理解する際に、全体的な同期の構造（どのプロセスとどのプロセスがどのイベントで同期しているのか）が、把握しづらいように感じる。このため、LOTOS記述を隅々まで読まないと、誤った理解をしてしまうことがあるように思う。プロセス定義の階層的記述に ある程度制限を加えるなど、全体的な同期の構造の把握が容易になるように記述スタイルを工夫する必要があるように思う。また、OSI各層に対するLOTOS記述の記述スタイルを確立し、LOTOS利用者に浸透させるような努力をしないと、このままではLOTOSは広まりにくいように感じる。

4.9.2 データ定義について

あるデータ型を定義しようとした場合、そのオペレータやオペレータの性質(等式)の適切な記述が容易に思いつかないことが多く、また 一応 記述したとしてもその記述が必要十分な記述であるのかという点については、自信の持てないことが多い。ただし、この点については、私が抽象データ型に慣れていないことが原因かもしれない。

LOTOS で OSI の応用層を記述しようとした場合、ASN.1定義をLOTOSのデータ定義で置き換える必要があり、これをそのまま忠実に記述しようとすると記述量がかなり膨大なものになる。これについては、ASN.1の各データ型に対するライブラリを作成しておいて、そのライブラリをうまく利用することによって記述量を減らすなどの工夫が必要であると考える。

4.9.3 その他

OSI各層のサービス定義やプロトコル仕様の記述にLOTOSのような形式記述言語を導入するのは、規約の記述においてあい昧性を排除するというだけでなく、プロトコルの検証やテストスイート生成の自動化に発展する可能性があるという意味において重要であると考える。しかし、現在自然言語で記述されている（ASN.1 も一部使用されている。）OSI各層の規約をLOTOSで記述し、それを実際の通信システムの開発現場で利用されるようにするのは、上記に述べた理由により容易でないと思われる。そのため、まず、適切な記述スタイルの確立やライブラリの整備などに力を注ぎ、実際の開発現場で利用されるようにする（または、利用できるものであるかどうかを見極める）必要があると考える。

4.10 Iさんの感想

4.10.1 状態遷移表との関係

状態遷移表は、通信プロトコルをプログラム言語で記述する際に大いに活用されるが、 Estelle はこの状態遷移表の形式を標準化しようとしているように見える。そのため状態遷移表に親しんでいる人にとってその記述は比較的理解しやすい。

LOTOSの場合を考えてみる。LOTOSにおいても抽象データ型の部分をうまく利用して状態遷移表のような記述が可能であると思う。[15]では、プロトコルの仕様を代数的言語OBJ2により形式的に記述しようという試みている。OBJ2は抽象データ型のみによって構成されるような言語である。抽象データ型の定義とはデータ対象とそれに対する操作とその結果の意味付けを定義することと理解しているが、[15]においてはそれはちょうど状態と入力事象と遷移先状態に対応させることができるということである。同じような手法によりLOTOSにおいてもそのような記述も可能であろう。

4.10.2 ソフトウェアの開発作業とLOTOS

LOTOSを記述することを考えると、データの定義の部分でどれだけのことをやり、プロセス定義の部分でどれだけのことをやるのかという配分がかなりの範囲で自由に選択できるような気がする。FDTのガイドライン [7] の中に、「ソフトウェアの開発作業は、ある抽象度を持つ仕様をより具象的な抽象度を持つ別のレベルの仕様にリファインすることで

ある」というような記述があるが、それはこのLOTOS記述の選択の自由さに一番良く適合するように思われる。

LOTOSは次のように使用されるのではないだろうか。まず、規格を元に抽象度の高い記述を行なう。これの後にはこれしか起こりません、データは大体こんな感じです、とかいったもの。そして、それに自然言語のコメントを沢山つける。その記述を少しずつ具象化していき、自然言語のコメントの意味あいを変えていく。最初は元の規格の記述のコピーであり、それをLOTOS文の記述を理解する上でのコメントとしていく。そしてあるレベル（どのレベルなのか良く分からないが）に達したら処理系に掛ける。

LOTOSに初めて触れたからかも知れないが、カッチリとしたLOTOSの記述で最初から仕様を考えていくというのは考えにくい様な気がする。

5 今後の検討課題

LOTOSのプログラミングスタイルについては文献[7]に4つの例題を用いて示されているが、必ずしも確立しているようには見えない。今後当研究会でも十分に現実のプロトコルをLOTOSで記述してみてその記述方法を叩いてプログラミングスタイルについて検討を積み重ねたい。

LOTOSの言語仕様も検討の必要がある。ASCEの記述を行なってみて例えばプロセスのスコープの概念とかゲートの動的割り付け等の機能も必要であると感じた。あるいはオブジェクト指向の概念を導入しコンパクトに記述できる様にもしたい。現在のLOTOSの仕様では、簡単なプロトコルでさえもそのLOTOSによる記述が非常に大きくなってしまう。

LOTOSの「処理系」とは何を指すのかに関して研究会の中では必ずしも議論が煮詰まってはいない。full LOTOSのコンパイルは不可能であることは容易に分かるので、工学的な観点から（つまり実働化するという観点から）いかにLOTOSの言語仕様をせばめ、計算機で処理可能な言語仕様にまで落すかというところが今後の検討課題である。

謝辞

LOTOS研究会は多くの方々の御協力により開催している。各社の関係者の方々には、LOTOS研究会に対して快くメンバを出していただいている。（財）情報処理相互運用技術協会には会議室使用について便宜を図っていただいている。電総研情報アーキテクチャ部棟上昭男部長および同部言語システム研究室二木厚吉室長にはLOTOS研究会に対して側面から支援をいただいている。ここに記して感謝する。

参考文献

[1] ISO 8807:"Information processing systems - Open System Interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behaviour", 1989-02-15.

[2] "Software Environment for Design of Open Dstributed Systems, HIPPO-LOTOS Simulator", the University of Twente, The Netherlands.

[3] 高橋、白鳥、野口:"LOTOS解釈機構の構成"、電子情報通信学会研究会、IN87-107、1988年1月22日.

[4] 神長、高橋、白鳥、野口:"LOTOS仕様の等価性とその判定法"、電子情報通信学会誌、J72-D-I、No.5、1989.

[5] 野村、長谷川、瀧:"LOTOS実行系の並列処理環境"、電子情報通信学会研究会、COMP89-4、1989年5月18日.

[6] 大蒔、二木:"図書館の問題とエレベータの問題のLOTOSによる仕様記述"、情報処理学会、ソフトウェア工学研究会、64-12、1989年2月2日.

[7] ISO/IEC JTC1/SC21 N3252: "Information Processing Systems - Open System Interconnection - Proposed Draft Technical Report on Guidelines for the Applicaton of Estelle, LOTOS, and SDL", 1989-06-01.

[8] International Symposium on Protocol Specification, Testing, and Verification, 1987, 1988など.

[9] P.H.J. van Eijk, C.A.Vissers, M.Diaz eds.: "The Formal Description Technique LOTOS", North-Holland, 1989.

[10] R.Milner:"A Calculus of Communicating Systems", Lecture Notes in Computer Science, 92, Springer-Verlag, 1980.

[11] 堀田:"型付きラムダ計算によるCCSの定式化とその効率のよい実行系"、電子情報通信学会研究会、COMP88-17、1988.

[12] P.H.J. van Eijk:"Software tools for the specification language LOTOS", Doctoral Dissertation, the University of Twente, 1988.

[13] M.Ben-Ari:"Principles of Concurrent Programming", Prentice-Hall Inc., 1982.

[14] C.A.R.Hoare:"Communicating Sequential Processes", Prentice-Hall Inc., 1985.

[15] K.Okada and K.Futatsugi:"Supporting the Formal Description Process for Communication Protocols by an Algebraic Specification Language OBJ2", Proc. of 2nd International Symposium on Interoperable Information Systems, pp.127-134, 1988.