

解説

LOTOS 言語の特質と処理系の現状と
動向†

高橋 薫†† 神長 裕明†††
白鳥 則郎††

1. はじめに

システムの仕様化は、システム開発における基本的な重要な要件であり、仕様化に際しては、厳密で正確な記述を可能とする形式記述技法 (FDT: Formal Description Technique) が不可欠である。FDT とは、形式的な構文および意味を備えた言語を用いて対象のシステムを定義する方法を指し、その具備条件としては、(1) 対象となる仕様が曖昧なく的確に記述できること、(2) 記述された仕様の適切な解析を可能とする形式的な数学モデルを提供すること、(3) 仕様の構造化・抽象化のための手段をもつこと、などがある。

LOTOS (Language Of Temporal Ordering Specification)^{[ISO 89a], [BOLO 87a], [TAKA 87]} はこのような FDT の一つとして、1981 年から 1988 年にかけて ISO により開発され、現在、国際標準として勧告されている。LOTOS は、特に、OSI アーキテクチャにおける各層のサービス定義やプロトコル仕様を形式的に記述することを目的としているが、一般的には、分散型の情報処理システムへの適用が可能である。

本稿では、まず前半で、LOTOS の言語的な機能について、例を主体とした解説を行う。紙面の都合上、すべてを網羅することは不可能であり、言語の構文・意味などについての詳細に興味のある読者は、末尾に示した関連文献を参照されたい。後半では、仕様の解析技術などの LOTOS に関係する支援技術と処理系の現状と動向について述べる。

2. 基本概念

LOTOS は基本的に、以下の考え方に基いて言語

設計が行われている。

『システムは、そのシステムとやりとりする外部から観測可能なイベント間の時間順序を規定することにより仕様化可能である』

LOTOS では、システムの記述を以下で述べる『プロセス』の観点から行い、外部から観測される振る舞いとしての通信能力を記述することでシステムの仕様記述を行う。プロセスの中で出現するデータの定義には、データ型の等式を用いた記述 (抽象データ型の代数的仕様記述) が使用される。上記のプロセスの振る舞い (動作) の記述は、R. Milner の CCS^[MILN 80] に基礎を置いており、記述された仕様の解析に有効な理論的枠組みを提供する。データの部分は抽象データ型^[INAG 86] の記述言語 ACTONE に基づいている。

『プロセス』とは、その環境 (対象としているプロセスの外側全体) における他のプロセスと通信を行う抽象的な実体を指す。プロセス間の通信は『ゲート』と呼ばれる作用点で起こり、通常、そこでデータ値の交換が行われる。インタラクション (通信による相互作用) の基本単位は『アクション』 (あるいは、『イベント』) と呼ばれ、ゲートとデータ値の組からなる。たとえば、あるゲート (場所) でのあるデータ値の交換 (送信や受信) が一つのアクションに対応する。アクションは『アトミック』で『同期的』なインタラクションである。アクションがアトミックであるというのは、(1) 瞬時発生であり、(2) 細分化できず、(3) 他のアクションと時間的な重なりがない、ということの意味する。アクションが同期的であるというのは、環境も含めて、通信する二つ以上のプロセスが、同時にその発生に関与するということを表す。

ゲートでのデータ値の交換をとみなない通信においては、ゲートのみがアクションを構成することになり、そのゲートのところで単にインタラクションが発生すると考えることができ、これはプロセス間の同期のみを表すことになる。

上記の概念によれば、典型的なプロセス P の静的側

† LOTOS Features with Survey of Their Support Processing Systems by Kaoru TAKAHASHI (Research Institute of Electrical Communication, Tohoku University), Hiroaki KAMINAGA (Computer Center, Tohoku University) and Norio SHIRATORI (Research Institute of Electrical Communication, Tohoku University).

†† 東北大学電気通信研究所

††† 東北大学大型計算機センター

面は図-1のようなブラックボックスとして表現することができる。同図において、 a, b, c はプロセス P が環境と通信するゲートを表している。LOTOS において、プロセス P の定義は次のようになる。

```
process P[a, b, c](...)... :=
    <behaviour-expression>
```

endproc

ここで、“process” および “endproc” はプロセス定義の始まりと終わりを示すキーワードである。これに対応して、キーワード “specification” および “endspec” があり、これらはトップレベルのプロセス（つまり、仕様自身）の定義の始まりと終わりを示す。かぎ括弧の中は仮ゲート引数、括弧の中は仮変数引数であり、通常のプログラミング言語に類似した役割を果たす（後述するプロセスインスタンス化の項を参照）。

プロセス P の動作は、図-1の例では三つのゲート a, b, c で起こる観測可能なアクションの系列を **behaviour-expression**（以下、“動作式”と略記）として記述することによって定義される。簡単化のため、 P は環境とのデータ交換をとまなわれないプロセスと考えよう。上述したように、この場合、アクションの構成要素はゲートのみとなり、そのゲートが関与する純粋な同期アクションを表す。このようなアクションについては、表記上、単にゲート名を用いて

a, b, c

のように表す。本稿の後半で触れるように、データ値を巻き込むアクションについては、

$q < v >$

のように表す。ここで、 q は関与するゲートであり、 v は交換されるデータ値である。

アクションの系列は、『木』を用いて表現できる。図-2に、このような『アクション木』の例を与える。この木は次のことを意味する。木の根から見ていくと、まず、枝のラベルとして示される a に出会う。こ

れは、最初にアクション a が環境と同期して起こりうるということを表したものである。次に、枝分かれが存在し、一方は b がラベルであり、もう一方は c がラベルである。これは、アクション b またはアクション c が環境と同期して生じうるということを言っている。 b が起こるか c が起こるかは環境に依存し、環境が b を起こすならば b が生起する。 c を起こすならば c が生起する。両方が生起する場合には、どちらが選択されるかは決まらない（非決定的）。いずれにしても、全体として、このアクション木は、最初にアクション a が生起可能であり、続いて、 b または c が生起可能であり、そして、終結するようなプロセスを表している。

プロセス P の動作が、このアクション木によって定義されるならば、その動作式は次のようになる（後述するように、実際にはもっといろいろな記述の仕方が存在する）。

$a; (b; \text{stop} \mid c; \text{stop})$

アクション木との対応は明らかであり、アクションの順次性はオペレータ “;”，分岐・選択はオペレータ “[]”，終結は “stop” により表される。

動作式を用いて記述された LOTOS プロセスの動的な意味は、次のように表記される『ラベル付き遷移 (labelled transition)』の集まりである『ラベル付き遷移システム (LTS: Labelled Transition System)』の観点から与えられる。LTS はグラフ構造を成し、それを『開く (unfold)』ことにより、前記のアクション木と対応する。

$B \xrightarrow{\text{act}} B'$ （ラベル付き遷移）

ここで、 B と B' は動作式であり、act はアクションである（前述のように、act はゲートのみかゲートとデータ値の組となる）。これを言葉でいえば、『プロセスの動作式 B はアクション act を実行し B' になる』あるいは『 B はアクション act の生起後 B' になる』となる。このようなラベル付き遷移（結果的には、LTS）を系統的に導出するため、遷移の公理と推論規則からなる遷移導出体系が定義されている [ISO 89a]。

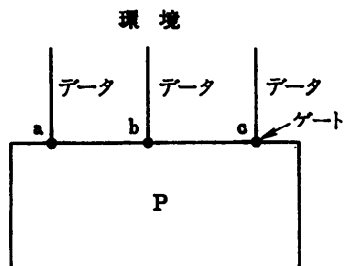


図-1 ブラックボックスとしてのプロセス

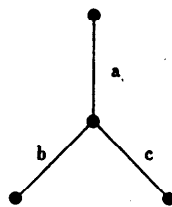


図-2 アクション木

LTS は、与えられた動作式、すなわち、プロセス定義に公理と推論規則を順次適用していくことにより得られる。LTS の形式的な定義は次のとおりである。

『定義1』 ラベル付き遷移システム

LTS は4項組 $\langle S, \text{Act}, T, s_0 \rangle$ である。ここで、 S は状態の集合、 Act はアクションの集合、

$$T = \{-\text{act} \rightarrow | -\text{act} \rightarrow \subseteq S \times S, \text{act} \in \text{Act}\}$$

は遷移関係の集合、 $s_0 \in S$ は初期状態である。□

与えられた LOTOS 動作式は LTS の初期状態に対応し、それから導出されたラベル付き遷移の動作式およびアクションが、それぞれ、状態集合およびアクション集合に対応する。ラベル付き遷移はまさに、遷移関係と対応する。 $(\text{cur}, \text{next}) \in -\text{act} \rightarrow$ がアクション act に対応する遷移関係（『定義1』に示すようにこれは $-\text{act} \rightarrow$ と表記され、状態対の集まりにより構成される）の要素であるとき、つまり $(\text{cur}, \text{next}) \in -\text{act} \rightarrow$ のとき、これがアクション act の生起による状態 cur から状態 next への遷移であることを明瞭に表すために $\text{cur} \rightarrow \text{next}$ と書く。

3. 言語機能

本章では、具体的な例をとおして、LOTOS の構文および意味を解説する。まず、3.1 ではいわゆる基本 LOTOS について述べる。これは、プロセス間の同期のみを取り扱う LOTOS である。3.2 では、3.3 で述べるフル LOTOS において使用されるデータ構造やデータ値の定義法について紹介する。フル LOTOS は、同期のみならず、プロセス間でのデータ値の交換を取り扱う LOTOS である。最後に、トランスポートプロトコルの仕様記述例（の一部）を与える。

3.1 基本 LOTOS

3.1.1 基本プロセス

どんなアクションも生起しない非アクティブなプロセスは、動作式

stop

によって表現される。したがって、これに関連した公理・推論規則はない。

3.1.2 基本オペレータ

(1) アクション・プレフィクス

B が既存の動作式で、a がアクション（基本 LOTOS の場合、ゲート名 a によって記述される純粋な同期アクション）のとき、

a; B

と書くことにより、『プロセスがアクション a を生起することができ、次に B によって表現される動作を行うことができる』ということを表す。アクションは内部アクション “i” であってもよい。内部アクションは、環境の介在なしにプロセス内部で自発的に起こりうるアクションであり、上述したような他の観測可能アクションとは区別される。

対応する公理を表-1 に示す。それは、『動作式 a; B で表されるプロセスは、アクション a の生起後 B になる、あるいは、a の生起後プロセスの動作は B によって表される』というように解釈される。

なお、表-1 は、後述する構文の意味も併せて示しており、これは、LOTOS 勧告 [ISO 899a] 中の意味定義の簡易版となっている。

(2) チョイス

B1 と B2 が既存の動作式のとき、

B1 [] B2

は、プロセスが B1 もしくは B2 のいずれかのように動作することを表す。どちらが選択されるかは、プロセスと環境との間の交信によって決まる。もし環境が B1 の最初のアクションを起こすならば B1 が選択される。環境が B2 の最初のアクションを起こすならば B2 が選択される。しかし、B1 と B2 の両方の最初のアクションが同時に起きるならば結果は決定されない。

対応する推論規則は表-1 のとおりであり、つまり、『B1 がアクション a の生起後 B1' になるならば、B1 [] B2 は a の生起後 B1' になる』および『B2 がアクション a の生起後 B2' になるならば、B1 [] B2 は a の生起後 B2' になる』というように解釈される。

〈例題〉 二つの端点間で全二重でデータを1回だけ転送するプロセスを考える（図-3 参照）。この一対のバッファの観測可能な動作を LOTOS で記述すると次のようになる。

```
process duplex_buffer[in1, in2, out1, out2] :=
    in1; ( in2; ( out1; out2; stop
              [ ] out2; out1; stop )
          [ ] out1; in2; out2; stop )
    [ ] in2; ( in1; ( out2; out1; stop
              [ ] out1; out2; stop )
          [ ] out2; in1; out1; stop )
endproc
```

この記述に、公理と推論規則を適用して得られるアクションは図-3 のようになる。

表-1 基本 LOTOS の意味

(action-prefix)	$a; B-a \rightarrow B$	
(choice)	$\frac{B1-a \rightarrow B1'}{B1[]B2-a \rightarrow B1'} \quad \frac{B2-a \rightarrow B2'}{B1[]B2-a \rightarrow B2'}$	
(process instantiation)	$\frac{Bp[q_1/q_1', \dots, q_n/q_n']-a \rightarrow B'}{p[q_1, \dots, q_n]-a \rightarrow B'}$	ただし, "process $p[q_1', \dots, q_n'] := Bp$ endproc" をプロセス p の定義とする
(relabelling)	$\frac{B-a \rightarrow B'}{B[q_1/q_1', \dots, q_n/q_n']-a' \rightarrow B'[q_1/q_1', \dots, q_n/q_n']}$	ただし, $a \in \{q_1', \dots, q_n'\}$ のとき $a' = a$. $a = q_i'$ ($1 \leq i \leq n$) のとき $a' = q_i$.
(parallel composition)	$\frac{B1-a \rightarrow B1', a \in \{q_1, \dots, q_n, \delta\}}{B1[]B2-a \rightarrow B1'[q_1, \dots, q_n]B2} \quad \frac{B2-a \rightarrow B2', a \in \{q_1, \dots, q_n, \delta\}}{B1[]B2-a \rightarrow B1[q_1, \dots, q_n]B2'}$ $\frac{B1-a \rightarrow B1', B2-a \rightarrow B2', a \in \{q_1, \dots, q_n, \delta\}}{B1[]B2-a \rightarrow B1'[q_1, \dots, q_n]B2'}$ $\frac{B1[]B2-a \rightarrow B'}{B1 B2-a \rightarrow B'} \quad \frac{B1[G]B2-a \rightarrow B'}{B1 B2-a \rightarrow B'}$	ただし, G はすべてのイベント (ゲート) のリスト
(hiding)	$\frac{B-a \rightarrow B', a \in \{q_1, \dots, q_n\}}{\text{hide } q_1, \dots, q_n \text{ in } B-a \rightarrow \text{hide } q_1, \dots, q_n \text{ in } B'}$ $\frac{B-a \rightarrow B', a \in \{q_1, \dots, q_n\}}{\text{hide } q_1, \dots, q_n \text{ in } B-i \rightarrow \text{hide } q_1, \dots, q_n \text{ in } B'}$	
(successful termination)	$\text{exit}-\delta \rightarrow \text{stop}$	
(enabling)	$\frac{B1-a \rightarrow B1', a \neq \delta}{B1 \gg B2-a \rightarrow B1' \gg B2} \quad \frac{B1-\delta \rightarrow B1'}{B1 \gg B2-i \rightarrow B2}$	
(disabling)	$\frac{B1-a \rightarrow B1', a \neq \delta}{B1 [> B2-a \rightarrow B1' [> B2} \quad \frac{B1-\delta \rightarrow B1'}{B1 [> B2-\delta \rightarrow B1'}$ $\frac{B2-a \rightarrow B2'}{B1 [> B2-a \rightarrow B2'}$	

3.1.3 プロセス・インスタネーション

プロセス・インスタネーション " $p[q_1, \dots, q_n]$ " は, プロセス名 p と実ゲート引数リスト $q_1, \dots, q_n$ からなり, 自分自身あるいは他で定義されたプロセス p をインスタネートする。インスタネートされたプロセスの動作は, その対応するプロセス定義によって与えられる。しかしながら, インスタネーションの際の実ゲートに対応する仮ゲートを巻き込むアクションは, その実ゲートを巻き込むアクションの発生へと置き換えて解釈される。これは『リラベリング (relabelling)』と呼ばれ, そのための推論規則が表-1 のように定義されている。

たとえば, 次のようなプロセス定義およびインスタネーションにより, "in; out; in; out; ..." の無限系列を表現できる (表-1 に従ってやってみるとよい)。

```

process buffer [in, out] :=
    in; buffer [out, in]
endproc

```

3.1.4 プロセスの並列性

(1) 同期しないプロセスの並列性

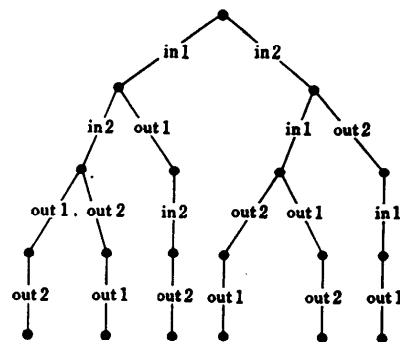
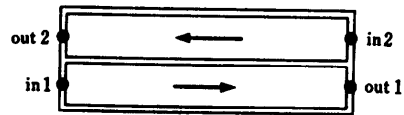


図-3 全二重バッファとそのアクション木

なんら互いに同期関係のない二つのサブプロセスを考える。これらのサブプロセスが動作式 $B1$ と $B2$ によって与えられているとすると,

$B1 || B2$

はこれらの独立的な並列合成を表す。つまり, $B1$ で

アクションが起こりうる時、あるいは、B2 でアクションが起こりうる時、 $B1 \parallel B2$ でそのアクションが起こりうる。一方、B1 と B2 の両方によって拒否されるどんなアクションも $B1 \parallel B2$ で起きえない。

Ⅲ を用いることにより、3.1.2 で示した duplex-buffer を、より構造的にまたより簡潔に定義することができる。

```
process duplex_buffer[in 1; in 2; out 1; out 2]
:=
    one_time_buffer[in 1, out 1]
  Ⅲ one_time_buffer[in 2, out 2]
where
    process one_time_buffer[in, out] :=
        in; out; stop
    endproc
endproc
```

ここで、“where” はサブプロセス定義の存在を示すキーワードである。このように、プロセス定義は階層的に行うことができる。

このプロセスの意味、すなわち、導出されるアクション木は前のもの(図-3)と等しい。オペレータ Ⅲ は、合成の動作を、サブプロセスのアクションの任意のインターリーブ(さしはさみ)として定義する。ここで、同じサブプロセスに属するアクションは、そのサブプロセス内の他のアクションについては指定された順序を保存するが、他のサブプロセスにおけるアクションとは順不同でよい。並列性のこの解釈は、アクションのアトミック性の仮定に基づいている。

(2) 完全同期するプロセスの並列性

相互に依存するサブプロセスの並列合成を反映させるためのオペレータがある。サブプロセスが B1 と B2 によって与えられたとき、

$B1 \parallel B2$

で、これらの同期的な並列合成を表す。ここで、B1 と B2 はすべてのアクションに関して同期しなければならない。B1 と B2 の両方で同じアクションが生じ可能のときに限り、 $B1 \parallel B2$ でそのアクションが生じうる。B1 あるいは B2 によって拒否されるどんなアクションも生じ不能である。

(3) 一般的並列オペレータ

二つのサブプロセスが B1 と B2 によって与えられたとき、

$B1[[g_1, \dots, g_n]]B2$

は、B1 と B2 がゲート $g_1, \dots, g_n$ を巻き込むアク

ション(基本 LOTOS の場合、これらのアクションは $g_1, \dots, g_n$ のもので記述される純粋な同期アクション)に関して同期しなければならないような並列合成を表す。この場合、同期すべきアクションは、上のようにそれらが関与するゲートのリストとして陽に与えられ、他のオペレータの場合のように、暗黙的ではない(Ⅲにおいては、このリストは空であり、Ⅱにおいては、すべてのゲートのリストである)。

3.1.5 アクションの隠蔽

B が既存の動作式のとき、

$\text{hide } g_1, \dots, g_n \text{ in } B$

は、ゲート $g_1, \dots, g_n$ を巻き込むアクションが内部アクションとなった B の動作を表す。これらのアクションは観測可能ではなく、環境の介在なしに自発的に起こる。このオペレータ(hiding オペレータ)は複数プロセス間の(多重)同期を、仕様記述対象のシステム内部の動作として、環境から隠してしまうような応用にしばしば使われる。たとえば、3.4 に示す記述例を参照されたい。

3.1.6 その他

上で述べたもののほかに、①プロセスの正常終結(successful termination)、②プロセスの順次合成(sequential composition)、③割り込み(disabling)、を反映するオペレータがある。それぞれの構文および意味については表-1 を参照されたい。

3.2 データ型

LOTOS における値、値式、データ構造の表現は、抽象データ型(ADT: Abstract Data Type)の記述言語 ACT ONE に由来する。ADT は、互いに関連したデータを結合し、それらに関して演算の構文と意味を定義したものである。ACT ONE は、パラメータ型仕様および非パラメータ型仕様を書くための、代数的仕様記述法であり、構造的な仕様を生成するために、次のような特徴をもつ。

- ライブラリとしてあらかじめ定義されているデータ型の使用
- 仕様の結合
- 仕様の名前の付け替え
- 仕様のパラメタ化
- パラメータ型仕様の具現化(actualization)
- 演算とソートによる仕様の拡張

以下では、LOTOS データ型定義のための基本的概念およびもっとも基本的な記述法に焦点を絞り説明する。より複雑なもの、また、データ型定義の形式的意

味解釈については文献 [ISO 89a] を参照されたい。

抽象データ型を定義するためには、データキャリア(台)と演算の名前を定義することが必要である。データキャリアの名前は『ソート』と呼ばれる。データ型のソートと演算の組は、そのデータ型の『シグニチャ』と呼ばれる。シグニチャはデータ型の構文を定義する。

以下では自然数を例にとり、データ型の定義法を紹介していこう。自然数のデータ型を参照するための名前を `Nat_numbers` とする。`Nat_numbers` のシグニチャは、ソート `nat` と演算 `0`, `succ` からなり、`succ` は `nat` の一つの元に適用されて、その結果としてやはり `nat` の一つの元を与えるような演算とする。この演算 `succ` の導入意図は、`succ(x)` が x の次の数 $x+1$ を表すようにさせるということである。`0` は引数をもたない演算とし、その意図はもちろん、自然数のゼロを表すようにさせるということである(ここでは、自然数は0からはじまると考えている)。この0のような演算は『定数』あるいは『リテラル』と呼ばれる。この場合、自然数の型定義は次のように与えられる。

```
type Nat_numbers is
  sorts nat
  opns 0      :      → nat
       succ : nat → nat
endtype
```

データキャリアのすべての元は、演算を繰り返し適用することによって得られる。そのような演算の適用の組み合わせの結果は『項』と呼ばれる。上のような自然数の仕様の場合、次のようなソート `nat` の項が生成される。

$0, \text{succ}(0), \text{succ}(\text{succ}(0)), \dots$

これらの項のそれぞれは、自然数の代数の一つの元と解釈することができ、シグニチャに関する情報を使って、代数のすべての元が表される。

さて、自然数上の計算を行うため、プラス演算の導入を考えよう。

$_{+} : \text{nat}, \text{nat} \rightarrow \text{nat}$

このプラス演算を加えることによって、

$\text{succ}(0) + \text{succ}(\text{succ}(0))$

のような別の項が生成される。この新たな項を正確に解釈するためには、演算の性質を表現するための新たな構文要素が必要であり、これは『等式』と呼ばれる。表記を簡単にするために、

$\text{succ}(\text{succ}(\dots \text{succ}(0) \dots))$ (succ が n 回)

を、 $\text{succ}^n(0)$ で表そう。この表記と等式を用いると、プラス演算の意味は次のように表すことができる。

$\text{succ}^n(0) + \text{succ}^m(0) = \text{succ}^{(n+m)}(0)$

しかしこれでは、すべての n と m に対して無限個の等式を書く必要があり、したがって、ソート全体にわたって等式を定義するために、『変数』を導入する。たとえば、

$x + 3 = \text{succ}^3(x)$ 。

定数3を自然数に加えるということは、今や、演算 `succ` を3回呼び出すことによって解決される。構文的には、項 $\text{succ}(\text{succ}(\text{succ}(x)))$ の中の変数 x は、0項演算(0のような定数)と同じカテゴリに属する。

等式と変数をもつ項の導入により、演算の性質を適切に記述する準備ができた。これにより、プラス演算の正確な定義は次のように与えることができる。

```
eqns forall x, y : nat
  ofsort nat
  x + 0      = x ;
  x + succ(y) = succ(x + y);
```

等式を表すのに用いられている記号“=”は、反射律・対称律・推移律・代入律を満たす同値関係を表している。そのソートの項全体の中で、“=”によって関係づけられるものを等価であるべき項として類別し、それらは同じ値を表すと考えるということである。この詳細については、たとえば文献 [ISO 89 a-付録 E], [INAG 86] を参照されたい。

プラス演算により拡張された自然数の仕様は次のようになる。

```
type Extended_Nat_numbers is
  sorts nat
  opns 0      :      → nat
       succ : nat → nat
       _+_ : nat, nat → nat
  eqns forall x, y : nat
  ofsort nat
  x + 0      = x ;
  x + succ(y) = succ(x + y);
endtype
```

3.3 フル LOTOS

3.3.1 構造的アクション

フル LOTOS ではデータ値の発生に関わる属性のリストをゲートと組み合わせることにより、構造的なアクションを記述できる。属性としては、『値宣言』と『変数宣言』の二つがある。

値宣言は、“!”マークによって先行される項(値式)であり、変数を含んでもよい。

(例) $!(3+5), !(x+1), !\text{true}$

値宣言属性とゲート名 g を組み合わせ、その項が値 v を表現するならば、この記述はアクション $g\langle v \rangle$ を表す。たとえば、 $\text{tsap}!(3+5)$ はアクション $\text{tsap}\langle 8 \rangle$ を表す。

値宣言属性をもったアクション・プレフィクス “ $g!E; B$ ” の意味は、次のように定義される。

$g!E; B \rightarrow g\langle \text{value}(E) \rangle B$

ここで、 $\text{value}(E)$ は項 E を解釈した値である。

変数宣言は、“ $?x:t$ ” の形をもつ。ここで、 x は変数名であり、 t はそのソート識別子である。ソート識別子は、 x が動く値の領域を示す。

(例) $?x:\text{nat}, ?\text{text}:\text{string}$

ゲート名 g が変数宣言属性 $?x:t$ と組み合わせられたとき、 $g?x:t$ はアクションの集合、すなわち、 t によって示される値領域中のすべての値 v について、すべてのアクション $g\langle v \rangle$ の集合を記述する。したがって、たとえば、 $\text{tsap}? \text{max}:\text{nat}$ はアクションの集合 $\{\text{tsap}\langle 0 \rangle, \text{tsap}\langle 1 \rangle, \dots\}$ を記述する。

変数宣言属性をもったアクション・プレフィクス “ $g?x:t; B$ ” の意味は次のように定義される。

$g?x:t; B \rightarrow g\langle v \rangle B$

($\text{domain}(t)$ 中のすべての v について)

ここで $[v/x]B$ は、 B 中の x を v で置き換えた結果である。また、 $\text{domain}(t)$ は t によって示される値領域を指す。

上で述べた属性を用いてプロセス間の種々のインタラクションを定義することができる。二つのプロセス間のインタラクションは、両方のプロセスが一つ以上の同一のアクションを生起可能のときに起こることができる。表-2 は種々の可能性をリストしている。

3.3.2 条件構文

値の発生に制限を与える述語を、前記の構造的アクションの記述に組み合わせることができる。たとえば、

$\text{sap1}?x:\text{nat}[x<2]; \text{sap2}!x; \text{stop}$
 $\rightarrow \text{sap1}\langle 0 \rangle \rightarrow \text{sap2}!0; \text{stop}$

$\text{sap1}?x:\text{nat}[x<2]; \text{sap2}!x; \text{stop}$

$\rightarrow \text{sap1}\langle 1 \rangle \rightarrow \text{sap2}!1; \text{stop}$

は、 $\text{sap1}?x:\text{nat}[x<2]; \text{sap2}!x; \text{stop}$ のすべての可能なアクションである。

動作式に述語を組み合わせることができる。この場合の解釈は、述語が成立するならば、動作式によって記述された動作が可能であり、述語が成立しなければ、動作式全体が stop と等価であるということである。たとえば、

$[x>0] \rightarrow \text{sap1}x; \text{some_process} [\dots] (x, \dots)$

で、 $x=1$ のとき、以下と等価である。

$\text{sap1}1; \text{some_process} [\dots] (1, \dots)$

3.3.3 プロセス定義のパラメタ化

プログラミング言語における関数や手続きと同様に、プロセス定義をパラメタ化することができる。基本 LOTOS においては、これはゲート名に関してなされたが、フル LOTOS では変数も仮引数とすることができる。仮ゲート名はかき括弧を用いて、また、仮変数引数は括弧を用いて表現される。プロセスインスタンス化において、仮ゲート名は実ゲート名によってリベリングされ、変数は同じソートの項によって置き換えられる。

3.4 仕様記述例

図-4 にトランスポートプロトコル(クラス2)の LOTOS 仕様 [BOCH 88] の一部を示す。仕様はトランスポートサービスユーザおよびネットワーク層との通信のためのゲート TS および NS でパラメタ化される。また、TC (transport connection), NC (network connection) の集合によりパラメタ化される。プロトコルはプロトコル機構を遂行する仮想的実体であるトランスポートエンティティ (TP_entity) により仕様化される。TP_entity は、(1) TC に対する処理を遂行するプロセス、(2) TC の多重化に関するプロセス、(3) NC の管理を行うプロセス、(4) TC リファレンスの使用を制約するプロセス、の4つの機能単位の並列合成として構成される。これらのプロセスは、同図に示すようなゲートで同期し、そこでのアクションは環境から隠される。環境(トランスポートサービスユーザおよびネットワーク層)からは、前記の TS および NS

表-2 プロセス間のインタラクションの型

プロセスA	プロセスB	同期条件	作用の種類	効果
$g!E_1$ $g!E$ $g?x:t$	$g!E_2$ $g?x:t$ $g?v:u$	$\text{value}(E_1) = \text{value}(E_2)$ $\text{value}(E) \in \text{domain}(t)$ $t = u$	値のマッチング 値の引き渡し 値の生成	同期 同期時に、 $x = \text{value}(E)$ 同期時に、 $v \in \text{domain}(t)$ について、 $x = v = v$

だけが見える。図-4 では、必要なデータ型定義の一例として TC 確立指示サービスプリミティブ TCONind を示している。この型では、TCONind の作成の演算、そして、TCONind の構成物の取出を表す射影演算が定義され、これらはプロセス定義の対応する箇所で使用されることになる。

ここでは詳しく述べないが、OSI アーキテクチャ上の概念、たとえば、サービスアクセス点、プロトコルデータ単位、などを FDT 上にどのようにリンクさせるか（これは『アーキテクチャ意味論』と呼ばれる）についての検討があり、仕様を記述する上でのガイドラインや種類の FDT の比較として有用となる。これについては、たとえば、文献 [TURN 87], [ISO 88] を参照されたい。

```

specification simple_TP[TS, NS](tc_ids: TCid_set, nc_ids: NCid_set): noexit
  (* 各種アドレス、サービスプリミティブ、PDU、などのデータ定義 *)
type TCONind is T_address, options (* TC 確立指示 *)
sorts TCONind
opns make_TCONind : T_address, options → TCONind
      source_adr : TCONind → T_address
      proposed_options : TCONind → options
eqns forall x: T_address, y: options
ofsort T_address
      source_adr(make_TCONind(x, y)) = x;
ofsort options
      proposed_options(make_TCONind(x, y)) = y;
endtype
behaviour
  TP_entity[TS, NS](tc_ids, nc_ids)
where
process TP_entity[TS, NS](tc_ids: TCid_set, nc_ids: NCid_set): noexit :=
  hide pr, ps, a, t in
    AP_modules[TS, pr, ps, a, t](tc_ids) (* TC の動作 *)
    Map[NS, pr, ps, a, t] (* 多重化 *)
    NC_managers[a, t](nc_ids) (* NC の管理 *)
    Unique_refs[a, t]({ } of ref_set) (* TC リファレンスの管理 *)
    (* pr: Map から AP への PDU の送信. ps: AP から Map への PDU の送信. *)
    (* a: NC への TC の割当て. t: NC への TC の割当て解除. *)
    (* ゲート a, t ではすべてのプロセスが同期. pr, ps では AP と Map が同期. *)
where
  (* AP_modules, などのプロセス定義および TP_entity 内部のデータ定義 *)
endproc
endspec

```

図-4 トランスポートプロトコルの LOTOS 仕様

4. 支援技術と処理系

本章では LOTOS の主な支援技術について、そのツールとしての処理系の動向も含めて要約する。

4.1 解析技術

LOTOS 仕様の解析のため、『弱 bisimulation 等価 (weak bisimulation equivalence)』、『試験等価 (testing equivalence)』と呼ばれる二つの同値関係が定義されている [ISO 89a], [BOLO 87a], [BRIN 87], [NICO 84]。これらの関係は、LOTOS 動作式の形式的解釈である LTS (ラベル付き遷移システム...『定義 1』を参照) の上で定義され、外部観測上区別不能な動作を行う LTS (結果的には、それに対応する動作式で定義されたプロセス) を互に関係づける。弱 bisimulation 等価である二つの LTS (に対応する二つのプロセス) は試験等価 [NICO 84] でもある。これらの等価性の違いは、その定式化の違いにあり、どちらを用いるかは使用者に任せられる。この応用としては、ネットワーク・アーキテクチャにおける層のサービス定義とプロトコル仕様をそれぞれ LOTOS で仕様化し、それらが外部観測上区別不能な動作をするかを検証すること [CARC 86] な

どがある。弱 bisimulation 等価性および等価性判定を行うアルゴリズムについては [BOLO 87b], [KAMI 89], [SHIR 89] で検討されている。これらのアルゴリズムの詳細についてはここでは述べないが、興味のある読者はこれらの文献を参照されたい。また、処理系としては、Squiggles [BOLO 88] や Metis [KAWA 89] がある。

動作式 B1 と B2 によって表現される二つの LOTOS 仕様の等価性は、B1 と B2 それぞれに公理と推論規則 (表-1 参照) を適用して得られる LTS Sys1 と Sys2 の等価性によって定義される。以下では、弱 bisimulation 等価を例にとり、LOTOS における等価性を LTS に基づいて説明する。

『定義 2』 アクション系列の遷移関係

Sys = <S, Act, T, s₀₀> を LTS とする。

(1) $s, s' \in S, a_1, \dots, a_n \in \text{Act}$ とし、 t をアクションの系列 $a_1 \dots a_n$ としたとき、遷移関係をアクションの系列 t に対して自然に拡張した関係 $s \xrightarrow{t}$ を次のように定義する。 $s = s_0 - a_1 \rightarrow s_1, \dots, s_{n-1} - a_n \rightarrow s_n = s'$ で

あるような状態 $s_0, \dots, s_n \in S$ が存在するとき、これを $s \xrightarrow{\varepsilon} s'$ で表す。特に、 $n=0$ に対しては $s \xrightarrow{\varepsilon} s$ である。ここで、 ε は空系列を表す。

(2) $s, s' \in S, a_1, \dots, a_n \in \text{Act} - \{\tau\}$, t を観測可能なアクションの系列 $a_1 \dots a_n \in (\text{Act} - \{\tau\})^*$ とし、 i^k を k ($k \geq 0$) 個の内部アクションの系列としたとき、関係 $t \Rightarrow$ を次のように定義する。 $s \xrightarrow{i^0 a_1 i^{k_1} \dots a_n i^{k_n}} s'$ であるようなアクションの系列 $i^{k_0} a_1 i^{k_1} \dots a_n i^{k_n}$ が存在するとき、これを $s \xRightarrow{t} s'$ と表す。特に、 $s \xrightarrow{i^k} s'$ を $s \xRightarrow{\varepsilon} s'$ で表し、また、すべての s に対して $s \xRightarrow{\varepsilon} s$ とする。

関係 $t \Rightarrow$ を用いて、弱 bisimulation 関係を定義する。

『定義 3』 弱 bisimulation 関係

$\text{Sys } 1 = \langle S_1, \text{Act}, T_1, s_{01} \rangle$, $\text{Sys } 2 = \langle S_2, \text{Act}, T_2, s_{02} \rangle$ を任意の LTS とし、 $S = S_1 \cup S_2$ とする。状態の組 $(s_1, s_2) \in R$ と、任意の観測可能なアクションの系列 $t \in (\text{Act} - \{\tau\})^*$ に対して、次の条件 (a) と (b) が成り立つとき、関係 $R \subseteq S \times S$ を弱 bisimulation 関係という。

- (a) $s_1 \xRightarrow{t} s_1'$ である $s_1' \in S$ が存在するならば、 $s_2 \xRightarrow{t} s_2'$ で $(s_1', s_2') \in R$ である $s_2' \in S$ が存在する。
- (b) $s_2 \xRightarrow{t} s_2'$ である $s_2' \in S$ が存在するならば、 $s_1 \xRightarrow{t} s_1'$ で $(s_1', s_2') \in R$ である $s_1' \in S$ が存在する。

『定義 4』 状態の弱 bisimulation

二つの LTS $\text{Sys } 1 = \langle S_1, \text{Act}, T_1, s_{01} \rangle$, $\text{Sys } 2 = \langle S_2, \text{Act}, T_2, s_{02} \rangle$ において、状態の組 (s_1, s_2) を含むような弱 bisimulation 関係が存在するとき、状態 s_1 と s_2 は弱 bisimulation 等価であるといい $s_1 \approx s_2$ と書く。

この定義において $\text{Sys } 1$ と $\text{Sys } 2$ は同一の LTS であってもよい。

『定義 5』 LTS の弱 bisimulation 等価

$\text{Sys } 1 = \langle S_1, \text{Act}, T_1, s_{01} \rangle$, $\text{Sys } 2 = \langle S_2, \text{Act}, T_2, s_{02} \rangle$ を任意の LTS とし、 $S = S_1 \cup S_2$ とする。 $(s_{01}, s_{02}) \in R$ であるような弱 bisimulation 関係 $R \subseteq S \times S$ が存在するとき (すなわち $s_{01} \approx s_{02}$ のとき)、 $\text{Sys } 1$ と $\text{Sys } 2$ は弱 bisimulation 等価であるといい、 $\text{Sys } 1 \approx \text{Sys } 2$ と書く。

動作式 (が表すプロセス) の等価性はその意味である LTS の等価性を用いて定義される。

『定義 6』 動作式の弱 bisimulation 等価

二つの動作式 B_1 と B_2 の LTS をそれぞれ $\text{Sys } 1$

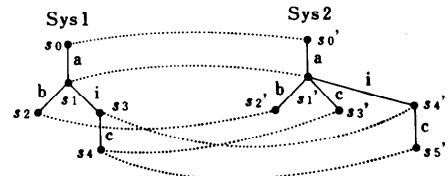
と $\text{Sys } 2$ とする。 $\text{Sys } 1 \approx \text{Sys } 2$ のとき、かつそのときに限り B_1 と B_2 は弱 bisimulation 等価であるといい、 $B_1 \approx B_2$ と書く。

図-5 に、弱 bisimulation 等価な動作式とその LTS、および等価ではない動作式とその LTS の例を示す。同図 (a) の場合、点線で示しているような状態間の関係 $R = \{(s_0, s_0'), (s_1, s_1'), (s_2, s_2'), (s_3, s_3'), (s_4, s_3'), (s_4, s_5')\}$ をとれば、これが弱 bisimulation 関係になっていることが分かる。このとき、初期状態の組 (s_0, s_0') は R の元であるから、二つの LTS $\text{Sys } 1$ と $\text{Sys } 2$ は弱 bisimulation 等価となり、よって二つの動作式 B_1 と B_2 は弱 bisimulation 等価となる。(b) の場合、 $s_0' = a \Rightarrow s_3'$ のとき、 $s_0 = a \Rightarrow s_1$ となる。初期状態の組 (s_0, s_0') を含む弱 bisimulation 関係が存在するためには、 s_3' と s_1 が弱 bisimulation 等価であることが必要である。しかし、 $s_1 = b \Rightarrow s_2$ であるにもかかわらず、 $s_3' = b \Rightarrow x$ となる状態 x が存在しないので s_3' と s_1 の等価性は成立しない。したがって、二つの LTS $\text{Sys } 1'$ と $\text{Sys } 2'$ は弱 bisimulation 等価とはならず、二つの動作式 B_1' と B_2' は等価ではない。

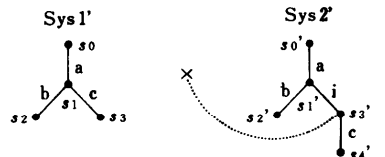
4.2 仕様の実行

仕様を実行する機構を用意することにより、仕様がどのような動的意味をもつか、あるいは、実装時にどのように動作するかを調べることができる。この目的のため、シミュレータが開発されている [ESPR], [GUIL 88] [TSUJ 89]。ここでは、東北大学で開発した LOTOS のシミュレータについて例を用いて述べる。

シミュレータは LOTOS 仕様を入力とし、LOTOS



$B_1 = a; (b; \text{stop}[\]i; c; \text{stop})$
 $B_2 = a; (b; \text{stop}[\]c; \text{stop}[\]i; c; \text{stop})$
 (a) 等価なプロセス



$B_1' = a; (b; \text{stop}[\]c; \text{stop})$
 $B_2' = a; (b; \text{stop}[\]i; c; \text{stop})$
 (b) 等価でないプロセス

図-5 弱 bisimulation 等価の例

の動的意味（遷移導出体系 [ISO 89 a]）に従ってその仕様の意味する動作を利用者主導でシミュレートする。シミュレーションの任意時点でするときに実行可能なアクションが導出され利用者に提示される。利用者は、実行可能アクションの中の一つを選択することによりそのアクションの実行を指示し、この操作の繰

り返しによりシミュレーションが行われていく。シミュレータの実行例を図-6に示す。この例は、コネクションの確立・データの転送・コネクションの切断を提供するサービス提供者を仕様化したものである（同図左下のウィンドウ）。これに対するシミュレーションの過程は同図右側のウィンドウに示されている。こ

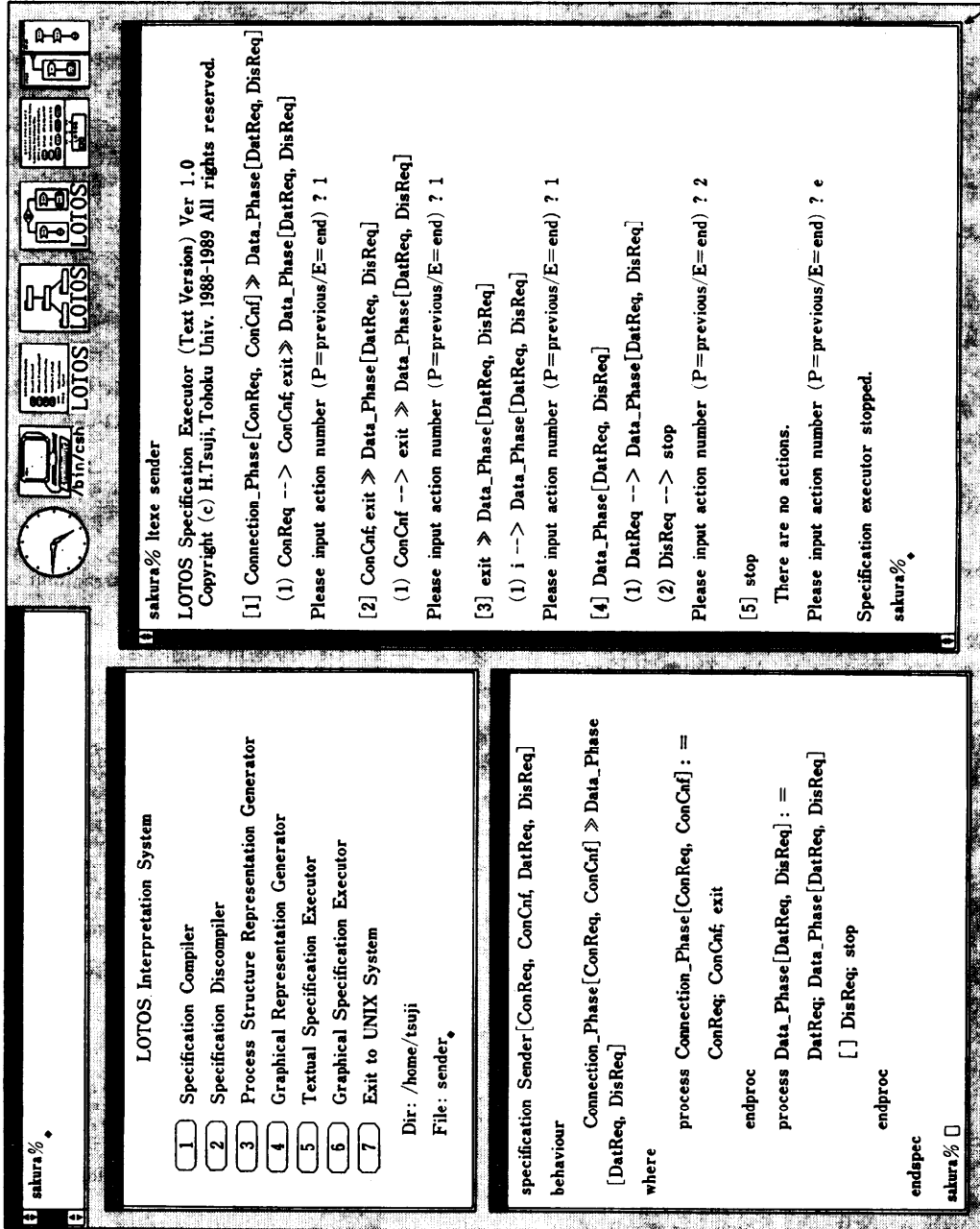


図-6 シミュレータの実行例

の例では、コネクションの確立要求からはじまって確立確認、そしてデータ転送は行わずただちに切断要求を起こして終了するような動作過程が利用者主導でシミュレートされている。

4.3 グラフ表現

現在、LOTOS はその記述形式としてテキスト表現が用いられている。しかしながら、記述された仕様の理解性や言語自身の学習の容易性を向上させるには、図を用いたグラフ表現を合わせて提供することが望ましい。ISO/CCITT では、現在、LOTOS のグラフ表現版 (G-LOTOS と呼ばれる)^[ISO 89b]を開発中であり、これにより LOTOS のよりいっそうの普及が期待される。なお、標準化機関ではなく、固有のグラフ表現も開発されており、テキスト表現からグラフ表現への自動変換や上記のシミュレータへの応用が行われている^[TSUJ 89]。

4.4 その他

他の支援技術として、(1) LOTOS の専用エディタ^[EIJK 89]、(2) 適合性試験のため、仕様からの試験系列の導出^{[MEER 87], [OKAZ 89]}、(3) 他の FDT への変換^[KARJ 88]、(4) 仕様の自動インプリメンテーション^[MANA 88]、などが検討されている。これらはいずれも、なお研究の初期段階にあり、今後の発展が望まれる。

5. おわりに

LOTOS は難しいといわれる。特に、仕様の読解性に難がある。筆者らの個人的見解ではあるが、これにはおおむね三つの要因があると考えられる。一つは、アクションの同期性にある。LOTOS において、内部アクションを除いたアクション(観測可能アクション)の発生にはすべて環境が介在する。このため、プロセス仕様を読む際には、アクションの発生に関して、常に同期関係にある他のプロセスおよびその仕様には陽に規定されない外部環境を考慮しなくてはならない。二つ目は、採用されている仕様記述のスタイルが次のように複数個あることである。本稿ではふれなかったが、LOTOS を用いた記述では、(a) システムのアクションの発生を種々の個別的な制約(たとえば、あるゲートに関与するアクションの時間的順序)の集まりとして表現する制約指向スタイル(constraint-oriented style)、(b) システムの状態を記述上に陽に反映させる状態指向スタイル(state-oriented style)、(c) 3.4 で示したように、システムの構成要素を記述上に

反映させるリソース指向スタイル(resource-oriented style)などの種々のスタイルがある^{[VISS 88], [TURN 88]}。

したがって、仕様を読む際には、しばしば、これらのおおののスタイルについての知識をあらかじめ必要とする。もう一つは、データ部にある。従来より親しんできたプログラミング言語におけるデータの記述法・表現法と LOTOS におけるそれとはかなりのギャップが存在するため、ある程度の慣れを必要とする。

本解説では LOTOS の言語的な特徴および支援技術について触れたが、あくまで、その一端にすぎない。これらのより詳細について興味のある読者は下記参考文献などを参照することを奨める。

謝辞 本稿の作成に協力をしていただいた東北大学野口研究室の諸氏に深謝します。

参 考 文 献

- [BOCH 88] Bochmann, G. V.: Specifications of a Simplified Transport Protocol Using Different Formal Description Techniques, Memorandum of Université de Montréal (1988).
- [BOLO 87 a] Bolognesi, T. and Brinksma, E.: Introduction to the ISO Specification Language LOTOS, Computer Networks and ISDN Systems, Vol. 14, pp. 25-59 (1987).
- [BOLO 87 b] Bolognesi, T. and Smolka, S. A.: Fundamental Results for the Verification of Observational Equivalence, Protocol Specification, Testing and Verification VI (1987).
- [BOLO 88] Bolognesi, T. and Caneve, M.: Squigles: a Tool for the Analysis of LOTOS Specifications, Proc. of FORTE 88, pp. 201-216 (1988).
- [BRIN 87] Brinksma, E., Scollo, G. and Steenberg, C.: LOTOS Specifications, Their Implementations and Their Tests, Protocol Specification, Testing and Verification VI, pp. 349-360 (1987).
- [CARC 86] Carchiolo, V. et al.: A LOTOS Specification of the PROWAY Highway Service, IEEE Trans. Computers, Vol. C-35, No. 11, pp. 949-968 (1986).
- [CLEA 89] Cleaveland, R. Parrow, J. and Steffen, B.: A Semantics based Verification Tool for Finite State Systems, Proc. of 9-th IFIP WG 6.1 International Symposium on Protocol Specification, Testing and Verification (1989).
- [EIJK 89] Eijk, P. V.: LOTOS Tools based on the Cornell Synthesizer Generator, Proc. of 9-th IFIP WG 6.1 International Symposium on Protocol Specification, Testing and Verification (1989).
- [ESPR] ESPRIT/SEDOS: HIPPO—LOTOS simulator, ESPRIT/SEDOS-ST 410.

- [GUIL 88] Guillemot, R., Haj-Hussein, M. and Logrippo, L.: Executing Large LOTOS Specifications, Protocol Specification, Testing and Verification **VII**, pp. 399-410 (1988).
- [HENN 88] Hennessy, M.: Algebraic Theory of Processes, MIT Press (1988).
- [HOAR 85] Hoare, C. A. R.: Communicating Sequential Processes, Prentice-Hall (1985).
- [INAG 86] 稲垣: 抽象データ型の概念と仕様記述法, 情報処理, Vol. 27, No. 2, pp. 120-128 (1986).
- [ISO 88] ISO: Guidelines for the Application of Estelle, LOTOS and SDL, ISO/IEC JTC 1/SC 21/WG 1 S 45 (1988).
- [ISO 89 a] ISO: Information Processing Systems-Open Systems Interconnection-LOTOS-A Formal Description Technique based on the Temporal Ordering of Observational Behaviour, ISO 8807 (1989).
- [ISO 89 b] ISO: G-LOTOS: A Graphical Syntax for LOTOS, ISO/IEC JTC 1/SC 21 N 3253 (1989).
- [KAMI 89] 神長, 高橋, 白鳥, 野口: LOTOS 仕様の等価性とその判定法, 電子情報通信学会論文誌, Vol. J 72-D-I, No. 5, pp. 367-376 (1989).
- [KARJ 88] Karjoth, G.: Implementing Process Algebra Specifications by State Machines, Protocol Specification, Testing and Verification **VII**, pp. 47-60 (1988).
- [KAWA 89] 川口, 神長, 高橋, 白鳥, 野口: LOTOS 仕様の等価性検証システムの構築, 信学技報, IN 89-59 (1989).
- [MANA 88] Mañas, J. A. and Miguel-More, T. D.: From LOTOS to C, Proc. of FORTE 88, pp. 79-84 (1988).
- [MEER 87] Meer, J. D.: Derivation and Validation of Test Scenarios based on the Formal Specification Language LOTOS, Protocol Specification, Testing and Verification **VI**, pp. 203-216 (1987).
- [MILN 80] Milner, R.: A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag (1980).
- [NAKA 89] 中原, 相田, 齊藤: シーケンス図を利用した LOTOS の視覚的デバッグ, 信学技報, IN 89-57 (1989).
- [NICO 84] Nicola, R. D. and Hennessy, M.: Testing Equivalences for Processes, Theoretical Computer Science, 34, pp. 83-133 (1984).
- [NOMU 89] 野村, ほか: LOTOS 実行系の並列処理環境, 情報処理学会ソフトウェア基礎論研究会資料, 89-4 (1989).
- [OHMA 89] 大蒔他: 哲学者の食事問題の LOTOS による記述実験, 情報処理学会ソフトウェア工学研究会資料, 66-4 (1989).
- [OKAZ 89] 岡崎, 高橋, 白鳥, 野口: LOTOS 仕様からの TTCN 表現によるテストシーケンスの自動生成, 信学技報, IN 89-25 (1989).
- [SHIR 89] Shiratori, N., Kaminaga, H., Takahashi, K. and Noguchi, S.: A Verification Method for LOTOS Specifications and Its Application, Proc. of 9-th IFIP WG 6.1 International Symposium on Protocol Specification, Testing and Verification (1989).
- [TAKA 87] 高橋, 白鳥: LOTOS および NESDEL の概要, 電子情報通信学会『交換・通信ソフトウェア仕様記述言語の標準化と技術展望』専門講習会資料 (1987).
- [TAKA 89] 高橋, 白鳥, 野口: LOTOS に基づいたプロトコル仕様の論理検証法, 信学技報, IN 89-26 (1989).
- [TSUJ 89] 辻, 高橋, 白鳥, 野口: LOTOS 解釈機構の設計と実現, 信学技報, IN 89-58 (1989).
- [TURN 87] Turner, K. J.: An Architectural Semantics for LOTOS, Protocol Specification, Testing and Verification **VII**, pp. 15-28 (1987).
- [TURN 88] Turner, K. J.: Constraint-Oriented Style in LOTOS, Memorandum of University of Stirling (1988).
- [VISS 88] Vissers, C. A., Scollo, G. and Sinderen, M. V.: Architecture and Specification Style in Formal Descriptions of Distributed Systems, Protocol Specification, Testing and Verification **VII**, pp. 189-204 (1988).

(平成元年9月1日受付)