ソフトウェア工学 87－5
(1992. 9. 29)

時制論理に基づくプロトコルのLOTOS仕様の合成

安藤敏彦[1], 加藤靖[1], 高橋薫[2], 野口正一[3]

[1]〒989-31, 仙台市青葉区上愛子北原1, 仙台電波工業高等専門学校
[2]〒980, 仙台市青葉区片平2-1-1, 東北大学電気通信研究所
[3]〒980, 仙台市青葉区片平2-1-1, 東北大学応用情報学研究センター

あらまし

形式記述技法の一つである**LOTOS**では安全性、生存性等の大局的な時間的性質を陽に記述することはできない。そこで本論文では、ラベル付き遷移システムから得られるアクション系列集合をモデルとする時制論理を用いることにより、時間的性質を陽に記述し、与えられた時間的性質から**LOTOS**仕様を導出する方法を提案する。この方法では、時間的性質を段階的に与えることで、段階的仕様化が達成でき、入れ子構造を持つプロセスの仕様を与えることができる。また、この方法のプロトコルへの適用により、あるクラスのプロトコル仕様が導出できることを示す。

和文キーワード　時制論理, **LOTOS**, プロトコル, ラベル付き遷移システム

Compositional LOTOS specification of protocol based on temporal logic

Toshihiko Ando[1], Yasushi Kato[1], Kaoru Takahashi[2], Shoichi Noguchi[3]

[1]Sendai national college of technology,
1, Kitahara, Kamiayashi, Aoba-ku, Sendai, 989-31 Japan
[2]Research Institute of Electrical Communication, Tohoku University,
2-1-1, Katahira, Aoba-ku, Sendai, 980 Japan
[3]Research Center for Applied Information Sciences, Tohoku University,
2-1-1, Katahira, Aoba-ku, Sendai, 980 Japan

Abstract

It is not easy to describe global temporal properties, such as safety property and/or liveness property, explicitly in **LOTOS**, one of the formal description techniques. So, we propose the compositional method that extracts a **LOTOS** specification from temporal properties. Temporal properties are described based on the temporal logic of which the model is a set of action sequences induced from a labelled transiiton system. By giving temporal properties stepwise, a stepwize development of specification can be achieved. We show an application of this method to a protocol.

英文 key words　temporal logic, LOTOS, protocol, labelled transition system

1 はじめに

プロトコル等、分散システムの仕様を記述する方法として形式記述技法 (Formal Description Technique, FDT) が用いられている。これは、自然言語に見られる曖昧さを避けるために開発されたもので、**LOTOS**、SDL、Estelle 等の FDT が国際標準化されている [1]。これら FDT を用いれば数学的に厳密な取り扱いが行なえる。ところが、どの FDT も個々のアクションに対するプロセスの振舞いを記述する形式を取るため、各時点での振舞いは示すことができても、時間的に離れたアクション間の関係等、プロセスの時間的な振舞いに関する全体的な性質を陽に記述することはできない。そのため、FDT で記述された仕様から時間的な性質を直接導出したり、直観的に理解することは困難である。

そのような FDT の局所性を補助する手段として、時制論理 (Temporal Logic) を利用することが考えられている [2]。時制論理は命題論理等の通常の論理体系に、□ (常に) 、◇ (いつか) 等の時間演算子を加え構成した論理体系である。時制論理は適用するモデルによって、線形時制論理 (Linear time Temporal Logic)、分岐時制論理 (Branching time Temporal Logic)[3] 等と呼ばれる。時制論理を用いれば、安全性 (safety property)、生存性 (liveness property) 等の性質を陽に示すことができ , 直観的に 理解しやすい 。時制論理を用いて FDT で記述された仕様が安全性、生存性を満足しているかどうかを検証する試みが行なわれている。例えば、有限状態機械と時制論理に基づいた SDL プロセスのモデルチェッカー [4]、**LOTOS** 仕様に対する時間的意味の構成的導出 [5] 等がそれである。また、これとは逆に、与えられた時間的性質を満たす仕様を導出することができるのであれば、仕様化の方法として有効である。しかし、一般に 1 つの時間的性質を満足する仕様は唯一ではなく、標準的な仕様の導出が望まれる。

本論文では、**LOTOS** 記述の意味であるラベル付き遷移システム (Labelled Transition System, LTS) から得られるアクション系列の集合をモデルとする拡張分岐時制論理 (Extended Branching Temporal Logic, EBTL) を定義し、EBTL で記述された時間的性質を基に、これを満足する基本 **LOTOS**(Basic LOTOS) 仕様の合成法を提案する。LTS は **LOTOS** 記述に対応しており、LTS が得られるならば、それに対応する **LOTOS** 記述が導出できる。さらに、この方法をプロトコルに適用することによって、あるクラスの **LOTOS** 仕様を得ることができることを示す。この方法は、時間的性質を順に追加していくことで、段階的な仕様化も達成可能となっている。

2 節では時制論理の概略および EBTL の定義を示す。また、3 節では時間的性質からの **LOTOS** 仕様の合成法を与える。そして、この方法のプロトコルへの適用例を 4 節で示す。

2 時制論理

状態系列 $\sigma=(s_0, s_1, s_2, \dots)$ 上で満足される性質を時制論理を用いて表現することができる [2, 3]。時制論理は、様相論理 (Modal Logic) の一種であり、様相論理で用いられる演算子□ (must)、◇ (may) を時系列上で解釈したものである。

2.1 線形時制論理

上で述べたように、時制論理は状態系列上で定義されることが多いが、**LOTOS** の場合には観測可能なアクション (イベント) を基にしているので、状態系列を扱うよりもアクションの系列を扱うことの方が重要である。ここで定義する線形時制論理では **LOTOS** 仕様の観測可能性に注目し、状態系列ではなくアクションの線形時系列をモデルとし、演算子□ (always)、◇ (sometime)、○ (next)、$\mathcal{U}$(until)、$\mathcal{W}$(unless) を用いる [5]。線形時制論理の構文を次のように定義する。

[定義 **1**] Φ を原子命題全体の集合とする。線形時制論理の論理式は次の規則によって構成される。

(1) $p \in \Phi$ ならば、p は論理式である 。

(2) P を論理式とすると、$\neg$P も論理式である 。

(3) P、Q を論理式とすると、$P \wedge Q$、$P \vee Q$、$P \Rightarrow Q$、$P \equiv Q$、$P \oplus Q$ は論理式である。

(4) P を論理式とすると、□P、◇P、○P は論理式である。

(5) P、Q を論理式とすると、$P\ \mathcal{U}\ Q$、$P\ \mathcal{W}\ Q$ は論理式である。

[]

上の定義において、(1) ～ (3) 命題論理の規則であり 、(4)、(5) が時間演算子に関わる規則である。

ここでは、アクションの系列をモデルとしているから、原子命題にはアクションが対応する。アクション全体の集合 *Act* は観測可能アクション、内部アクション i、正常終了アクション δ からなる。

[定義 **2**] 線形時制論理のモデルはアクション系列 $\sigma=(a_0, a_1, a_2, \dots)$ である 。ここで 、$a_i \in Act (i \in \mathbf{N}$ 、N は自然数全体の集合)、*Act* はアクション全体の集合である。 []

また、線形時制論理の論理式は、充足関係 (satisfaction relation) $\models$ によって、アクション系列上で次のように意味を与えられる。

[定義 **3**] アクション系列 $\sigma=(a_0, a_1, a_2, \dots)$ をモデル、P、Q を論理式、$a \in Act$ とすると、充足関係 $\models$ は σ の第 i アクションに対して次のように定義される 。

$\sigma, i \models$a iff a_i=a
$\sigma, i \models \neg$P iff not $\sigma, i \models$P
$\sigma, i \models$ P $\wedge$ Q iff $\sigma, i \models$P and $\sigma, i \models$Q
$\sigma, i \models \Box$P iff $\forall j \geq i, \sigma, j \models$P
$\sigma, i \models \Diamond$P iff $\exists j \geq i, \sigma, j \models$P
$\sigma, i \models \bigcirc$P iff $\sigma, i+1 \models$P
$\sigma, i \models$ P $\mathcal{U}$ Q iff $\exists j \geq i$, $\sigma, j \models$Q
and $i \leq \forall k < j$, $\sigma, k \models$P
$\sigma, i \models$ P $\mathcal{W}$ Q iff $\sigma, i \models$P $\mathcal{U}$ Q or $\sigma, i \models \Box$P []

また、その他の命題論理の演算子は次のように定義される。

(P$\vee$Q) $=_{DF}$ $\neg(\neg P \wedge \neg Q)$
(P$\Rightarrow$Q) $=_{DF}$ $\neg P \vee Q$
(P$\equiv$Q) $=_{DF}$ $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$
(P$\oplus$Q) $=_{DF}$ $(P \wedge \neg Q) \vee (\neg P \wedge Q)$

同様に、2つの時間演算子も各々他方によって表現できる。

$\Box$ P $=_{DF}$ $\neg\Diamond\neg$P
$\Diamond$ P $=_{DF}$ $\neg\Box\neg$P

2.2 拡張分岐時制論理

一般に、一つの **LOTOS** 仕様に対して複数のアクション系列が得られる。そのような系に対しては上の演算子だけでは不十分で、ここで定義する拡張分岐時制論理 (EBTL) で表現する必要がある [3]。これは、分岐するアクション系列をモデルとし、拡張時制論理の演算子を拡張したものである。EBTL の構文を次に示す。

[定義 4] Φ を原子命題全体の集合とする。EBTL の論理式は次の規則によって構成される。

(1) P を命題論理の論理式とすると、P は論理式である。

(2) P を論理式とすると、ALWAYS P、always P、SOMETIME P、sometime P、NEXT P、next P は論理式である。

(3) P、Q を論理式とすると、P UNTIL Q、P until Q、P UNLESS Q、P unless Q は論理式である。

[]

次に、EBTL の意味を示す。1つのアクション系列 σ =(a_0 , a_1, ...) の有限部分列 (a_0, a_1, ..., a_i) を先頭に持つ系列 (a_0, a_1, ..., a_i, b_{i+1} , ...) の集合を Seq_{σ}^{+i} と書き、σ から派生するアクション系列集合と呼ぶものとする。これは1つの LTS から得られるアクション系列の集合を想定している。この Seq_{σ}^{+i} が、EBTL のモデルとなる。この時、EBTL の意味は次のように定義される。

[定義 5] アクション系列 σ_0 から派生するアクション系列の集合の組 $\mathcal{M}_{\sigma_0} = \langle Seq_{\sigma_0}^{+0}, Seq_{\sigma_0}^{+1}, Seq_{\sigma_0}^{+2}, \ldots \rangle$ をモデル 、P、Q を論理式 、a$\in Act$ とすると、EBTL の充足関係 $\models$ は σ_0 の第 i アクションに対して、拡張時制論理の充足関係を用いて次のように定義される。

$\mathcal{M}_{\sigma_0}, i \models$a iff $\sigma_0, i \models$a
$\mathcal{M}_{\sigma_0}, i \models$P iff $\sigma_0, i \models$P
$\mathcal{M}_{\sigma_0}, i \models \neg$P iff $\sigma_0, i \models \neg$P
$\mathcal{M}_{\sigma_0}, i \models$ P $\wedge$ Q iff $\sigma_0, i \models$P and $\sigma_0, i \models$Q
$\mathcal{M}_{\sigma_0}, i \models$ALWAYS P iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Box$P
$\mathcal{M}_{\sigma_0}, i \models$always P iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Box$P
$\mathcal{M}_{\sigma_0}, i \models$SOMETIME P iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Diamond$P
$\mathcal{M}_{\sigma_0}, i \models$sometime P iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Diamond$P
$\mathcal{M}_{\sigma_0}, i \models$NEXT P iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \bigcirc$P
$\mathcal{M}_{\sigma_0}, i \models$next P iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \bigcirc$P
$\mathcal{M}_{\sigma_0}, i \models$P UNTIL Q iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P $\mathcal{U}$ Q
$\mathcal{M}_{\sigma_0}, i \models$P until Q iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P $\mathcal{U}$ Q
$\mathcal{M}_{\sigma_0}, i \models$P UNLESS Q iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P $\mathcal{W}$ Q
$\mathcal{M}_{\sigma_0}, i \models$P unless Q iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P $\mathcal{W}$ Q

ただし、命題論理の演算子は前述通り定義されるものとする。[]

今後、EBTL の論理式を時間的性質 (temporal property) と呼ぶことにする。また、時間的性質を記述するのに便利な演算子も定義しておく。

$\mathcal{M}_{\sigma_0}, i \models$FREQ P iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Box\Diamond$P
$\mathcal{M}_{\sigma_0}, i \models$freq P iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models \Box\Diamond$P
$\mathcal{M}_{\sigma_0}, i \models$P BEFORE Q iff $\forall \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P$\wedge \bigcirc\Diamond$Q
$\mathcal{M}_{\sigma_0}, i \models$P before Q iff $\exists \sigma \in Seq_{\sigma_0}^{+i}, \sigma, i \models$P$\wedge \bigcirc\Diamond$Q

これら演算子の内、大文字で書かれたものは全ての系列で真となることを示し、小文字で書かれたものは真となる系列が存在することを示す。

3 時間的性質からの LOTOS 仕様の導出

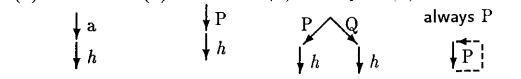
一般に、仕様に対する時間的性質は一意に決定できるが、与えられた時間的性質に対してそれを満足する仕様を一意に決定することはできない。しかし、対象とするプロセスが複数のブロックから構成できるようなものとすれば、ある程度仕様の構造が決定される。ここでは、与えられた時間的性質からそれを満足する **LOTOS** 仕様を導出する方法を示す。

3.1 時間演算子と LTS との対応

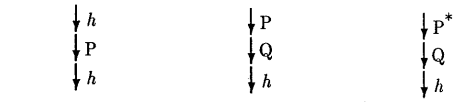
本節では、基本的な時間演算子に対する標準的な LTS を定める。ここで、標準的な LTS とは、与えられた時間的性質を満たし、アクション数が最小である LTS を言うものとする。これらは、LTS を合成するための部品として用いられる。

[定義 6] EBTL の演算子、および特定の論理式に対する標準的な LTS を次のように定義する。ここで、h はダミーアクションであり、空であるか、または他の LTS に置き換え得るものとする。a はアクション、P、Q は EBTL の論理式とする。また、$*$ はその性質を満たす有限なアクション系列であることを示す。

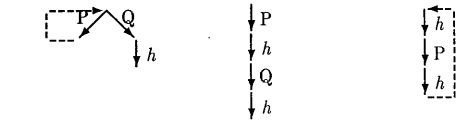
(1) a (2) P (3) P ⊕ Q (4) ALWAYS P, always P



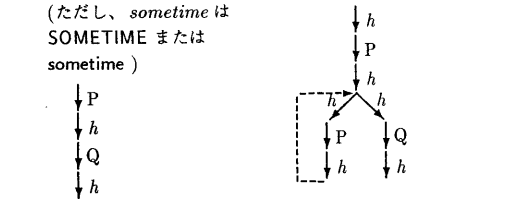
(5) SOMETIME P, sometime P (6) P ∧ NEXT Q , P ∧ next Q (7) P UNTIL Q, P until Q



(8) P UNLESS Q , P unless Q (9) P BEFORE Q, P before Q (10) FREQ P, freq P



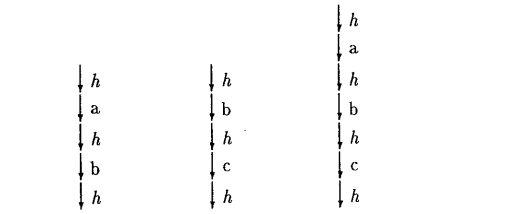
(11) P ⇒ (*sometime* Q) (ただし、*sometime* は SOMETIME または sometime) (12) (FREQ P) ⇒ (sometime Q)



[]

次の例は LTS の接続における時間的性質の保存を示すものである。

[例 1] 性質 P_1 : SOMETIME (a⇒(SOMETIME b))、P_2 : SOMETIME (b⇒(SOMETIME c)) に対する標準的な LTS を、両者に共通な観測可能アクション b を重ねるようにして接続しても、P_1、P_2 は保存される。



性質 P_1　　性質 P_2　　接続された LTS []

従って、例 1 から次の性質が考察される。

[性質 1] 2個の LTS をシーケンシャルに接続することができれば、各々の時間的性質は保存される。ここで LTS S_1、S_2 をシーケンシャルに接続するとは、S_1 のある終端を S_2 の根に接続することを言う。ただし、観測可能アクションを含む共通の連結部分 LTS があれば、新たな分岐が生じないようにそれを重ねるものとする (例 1 の場合、hbh)。 []

性質 1 は 2 つの時間的性質の ∧ に対応する LTS の合成法を示している。 すなわち、性質 1 は複数の時間的性質を満足するような LTS の合成の原理となる。

3.2 時間的性質からの LOTOS 仕様の導出

以下では、時間的性質から **LOTOS** 仕様を導出するための構成法を示す。ここで次の記法を用いる。LTS S にダミーアクション h を除き陽に含まれるアクションの集合を Act (S)、 S の根を含む連結な部分 LTS の全体を $Root$ (S) とする。また、時間的性質 P に h を除き陽に含まれるアクションの集合を Act (P) とする 。

[構成法 1] 時間的性質に基づく **LOTOS** 仕様の合成

(1) 性質 $P_1, \ldots, P_n$ を与える。P_1 は全プロセスの初期状態に関わるものとする 。

(2) 定義 6 に従って、$P_i (1 \leq i \leq n)$ を満足する標準的 LTS S_i を求める。ただし、Act (S_i) =Act (P_i) であるものとする。

(3) S_i に含まれる部分 LTS で $S_j (1 \leq j \leq n, j \neq i)$ にも含まれるものを $S_{(i,j)}$ とする 。ただし、Act ($S_{(i,j)}$)≠ $\emptyset$ 。ここで、$2 \leq i \leq n$ の S_i に対して、$\forall j \neq i$, $S_{(i,j)} \notin Root$ (S_i) であるならば (1) に戻り性質を与え直す。また、$S_{(i,j)}$=S_i または $S_{(i,j)}$=S_j である時も (1) に戻る。

(4) (3) で求めた $S_{(i,j)}$ が重なるよう、S_i と S_j とを接続する。

(5) プロセスの動作が有限ならば、未接続の終端すべてに、プロセスの終了を示す特別のアクション δ を付け加える。また、動作が無限ならば、終端を S_1 の根に接続する。

(6) ループに分岐する h を内部アクション **i** で置き換える。必要ならば、その他の h アクションを **i** で置き換えてもよい。最後に、残りの h アクションを消去する。ここで得られた LTS を S とする。S が求める LTS である 。

(7) LTS S を **LOTOS** 記述へ変換する。

[]

構成法 1 の (2) では、複数の小さな LTS をシーケンシャルに接続できるものだけを考えている。構成法 1 の (6) においてループに分岐する h(例えば、定義 6(12) に見ることができる) を内部アクションで置き換えるのは、このようなループがプロトコルにおいてはデータ等の再送を意味し、その原因は媒体の障害等観測不可能なアクションによると考えられるからである。

LTS から **LOTOS** 記述への変換法を変換法 1 に示す。

[変換法 1]　LTS から **LOTOS** 記述への変換

(1) 全体のプロセス名を $Proc_1$ とする。また、h アクションから置き換えられた内部アクションを、各々 $int_1, int_2, \ldots, int_p$ とし、合流地点に各々サブプロセス名 $Proc_2, \ldots, Proc_q$ をつける。

(2) **process**　$Proc_1$ [Δ] :=

を生成する。Δ には、Act (S) のアクションを"," で区切って列記する。根のアクションから合流地点までに内部アクションがあれば、次に

hide Δ_{int} **in**

を挿入する。ここで、Δ_{int} は合流地点までに出現する内部アクションのリストである。

(3) 根のアクションから順に次の手続きを行なう。

(3.1) アクション系列が分岐しなければ、一続きのアクションの間に";" (prefix) を挟み、アクションを列記する。

(3.2) アクションが分岐すれば、分岐してできるアクション系列を" []" (choice) で結ぶ。

(3.3) アクション系列が終端に達したら、それが δ アクションならば "**exit**"で置き換え、それ以外であれば、最後のアクションに "; **stop**"を付け加える。また、合流地点に達したら、"; (対応するサブプロセス名)" を付け加える。

(3.4) 最後に、

where
endproc

とする。

(4) 各サブプロセスは $Proc_1$ と同様に記述し、$Proc_1$ の **where** と **endproc** の間に挿入する。

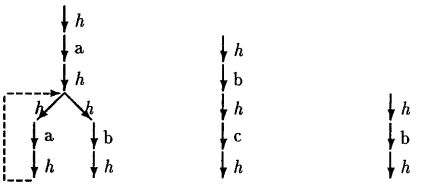
[]

構成法 1 の例を示す。

[例 **2**]　2 つの性質 P_1 : (freq a)⇒(sometime b)、P_2 : SOMETIME (b⇒(SOMETIME c)) を満足する LTS を合成する。

(1) 各時間的性質に対する LTS を求める (構成法 1 の (2))。

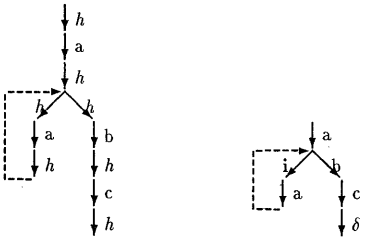
(2) 両者に共通する部分 LTS を求める (構成法 1 の (3))。



性質 P_1　　性質 P_2　　共通部分

(3) 2 つの LTS を接続する (構成法 1 の (4))。

(4) ループに分岐する h アクションを内部アクションに置き換え、その他の h アクションを消去する (構成法 1 の (5),(6))。



例 2 の (3)　　例 2 の (4)

(5) **LOTOS** 記述を求める (構成法 1 の (7) および変換法 1)。

```
process Proc1[a, b, c] :=
    a; Proc2[a, b, c]
where
    process Proc2[a, b, c]:=
        hide int1 in
            int1; a; Proc2[a, b, c]
        [] b; c; exit
    endproc
endproc
```

[]

3.3　段階的構成法

次に、構成法 1 を基本とした **LOTOS** 仕様の段階的な構成法を示す。より多くの時間的性質を付加し、構成法 1 の過程を繰り返せば、より詳細な仕様が得られる。すなわち、最初はプロセスの大まかな性質を与え、その後の段階では、より詳細な性質を与えるというように、段階的な仕様化を達成することができる。追加する性質は、前段階の性質に現れるアクションとアクションの間を補うように与える。

[構成法 2] 時間的性質に基づく **LOTOS** 仕様の段階的構成法

(1) 今、性質 $P_1^{(1)}, \ldots, P_n^{(1)}$ から、構成法 1 に従って、LTS $S^{(1)*}$ が構成されているものとする。ただし、ダミーアクション h を消去せず残してある。

(2) 性質 $P_1^{(1)}$ に対し、性質 $P_{1,1}^{(2)}, \ldots, P_{1,n_1}^{(2)}$ を与える。これらの性質は $P_1^{(1)}$ に現れるアクションとアクションの間を補うものとする。

(3) 性質 $P_{1,1}^{(2)}, \ldots, P_{1,n_1}^{(2)}$ を満たす標準的な LTS を求め、この LTS を $S_1^{(2)}$ とする。

(4) $S^{(1)*}$ において、$P_1^{(1)}$ に相当する LTS $S_1^{(1)}$ を $S_1^{(2)}$ に置き換える。

(5) $P_i^{(1)}(2 \leq i \leq n)$ について $P_1^{(1)}$ と同様に (2) ～ (4) を行なう。

(6) プロセスの動作が有限ならば 、 未接続の終端すべてに、プロセスの終了を示す特別のアクション δ を付け加える。また、動作が無限ならば、終端を $S^{(1)*}$ の根に接続する。ここで得られる LTS を $S^{(2)*}$ とする。

(7) 必要であれば 、$S^{(2)*}$ を $S^{(1)*}$ とみなして (1) ～ (6) を繰り返し、LTS $S^{(3)*}$ を求める。同様に、必要であれば、$S^{(4)*}$, $S^{(5)*}$, ... を求める。

(8) $S^{(p)*}$ $(p \geq 2)$ において、ループに分岐する h を内部アクション **i** で置き換える。必要ならば 、 その 他の h アクションを **i** で置き換えてもよい 。 最後に 、 残りの h アクションを消去する 。 このようにして得られた LTS を $S^{(p)}$ とする 。

(9) LTS $S^{(p)}$ $(p \geq 2)$ を **LOTOS** 記述に変換する。

[]

構成法 2 の (3) は、$S^{(p)*}$ に現れる h を追加した時間的性質に対する LTS に置き換える作業である。すなわち、1 つのサブプロセスをさらに小さなサブプロセスでシーケンシャルに構成することに相当する 。従って、構成法 2 は入れ子構造を持つプロセスの仕様化に有効である。また、構成法 2 を繰り返すことによって、レベルの違った LTS の列 $S^{(1)}, S^{(2)}, \ldots$ が得られるが、これらの LTS の関係を次の性質 2 に示す。

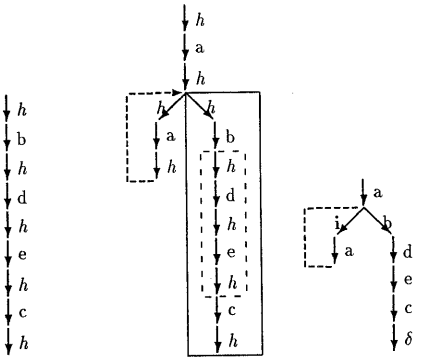
[性質 2] LTS S が満足する性質の集合を $Prop(S)$ とすると、$m < n$ ならば、$Prop(S^{(m)}) \subset Prop(S^{(n)})$ である。 []

[例 3] 例 2 で与えた時間的性質に、新たな性質を付加し、仕様をより詳細にする。付加する条件は、$P_{2,1}$: SOMETIME (b⇒ (SOMETIME d))、$P_{2,2}$: SOMETIME (d⇒ (SOMETIME e))、$P_{2,3}$: SOMETIME (e⇒ (SOMETIME c)) である。

(1) 付加する性質に対する標準的な LTS を求める。(構成法 2 の (3))

(2) 例 2 の (3) の LTS において、P_2 の LTS を上の LTS に置き換える (構成法 2 の (4))。実線四角内は置き換えられた部分、破線四角内は追加した性質によって補われた部分である。

(3) 分岐する h アクションを内部アクションに変え、有限動作であるから、終端に δ アクションを付け加え、余分な h アクションを消去する (構成法 2 の (6)、(8))。



例 3 の (1)　　例 3 の (2)　　例 3 の (3)

(4) この LTS を **LOTOS** 記述に変換する (構成法 2 の (9))。

```
process Proc1 [a, b, d, e, c] :=
    a; Proc2[a, b, d, e, c]
where
    process Proc2[a, b, d, e, c]:=
        hide int1 in
            int1; a; Proc2[a, b, d, e, c]
        [ ] b; d; e; c; exit
    endproc
endproc
```

[]

4 プロトコルへの適用

本節では、構成法 1、2 のプロトコルへの適用を行なう。ここでは、プロトコルの生存性を時間的性質として与え、全体の振舞いを示す LTS を得、そして対応する **LOTOS** 仕様を導出する。生存性とは「将来、良いことが起こるであろう」ことを示す性質である [2]。例えば 、

(freq (データを送る))⇒(sometime (データを受け取る))

: 送信側がデータを送り続ければ、いつか受信側はデータを受け取る。

ALWAYS ((データを受け取る)⇒(SOMETIME (Ack を送る)))
: 受信側がデータを受け取れば、受信側は送信側に Ack を送る。

がそうである。

4.1 アブラカダブラプロトコル

例として、送信側 (Sender) と受信側 (Receiver) の間で、信頼度の低い媒体 (Medium) を通した一方向通信のプロトコルを考える。これ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の3つのフェーズを持つものとする。これは, アブラカダブラプロトコル (Abracadabra Protocol) の簡略版である [1]。データ等が媒体で欠落する可能性があり、欠落した場合は再送で補うものとする 。

4.1.1 各フェーズの概要

(1) コネクション確立フェーズ (Connection Phase)

送信側が受信側に接続要求 (CR) を送り、送信側が受信側から接続確認 (CC) を受け取ると、コネクションが確立され、データ転送フェーズに移る。CR、CC の欠落は再送によって復旧させる。切断要求 (DR) の送信はいつでも可能であり、DR が送られると、コネクション切断フェーズに移る。

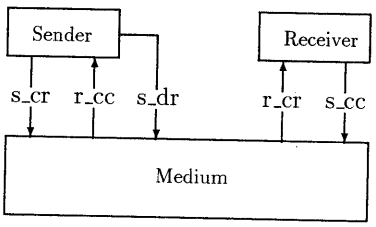


図 1 : コネクション確立フェーズの概念図

(2) データ転送フェーズ (Data Transfer Phase)

送信側が受信側にデータを転送 (DT) し、受信側は送信側に受信確認 (AK) を送る。DT、AK の欠落は再送によって復旧させる 。切断要求 DR の送信はいつでも可能であり、DR が送られると、切断フェーズに移る。

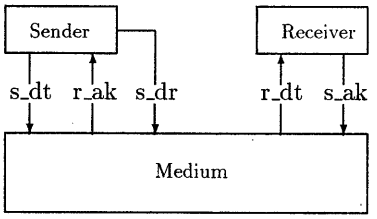


図 2 : データ転送フェーズの概念図

(3) コネクション切断フェーズ (Disconnection Phase)

送信側が受信側に切断要求 DR を送り、送信側が受信側から切断確認 (DC) を受け取ると、コネクションが切断される。その後、コネクション確立フェーズに移る 。

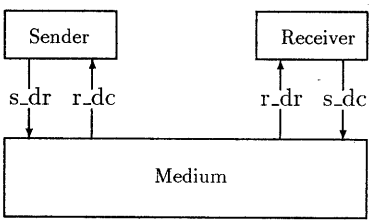


図 3 : コネクション切断フェーズの概念図

上の3つのフェーズは次のように関連づけられる。Idle は初期状態である。

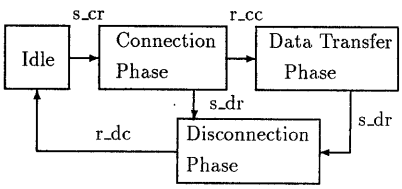


図 4 : 各フェーズ間の関係

4.1.2 プロセス全体としての性質

プロセス全体の性質として、フェーズの遷移に関わる性質を与える。これらは特定のアクションが起これば次のフェーズに移るであろうことを示しているので、生存性である。全体的な時間的性質を次に示す 。

(1) ALWAYS (s_cr ⇒ ((SOMETIME (r_cc ⊕ s_dr)))
: [コネクション確立フェーズ] 送信側が CR を送れば、いつか送信側は CC を受け取るか、または、DR を送る。

(2) ALWAYS (r_cc ⇒ (sometime s_dr))
: [データ転送フェーズ] 送信側が CC を受け取ればデータ転送フェーズに入り、送信側が DR を送るまではデータ転送フェーズに留まる。

(3) ALWAYS (s_dr ⇒ (SOMETIME r_dc))
: [コネクション切断フェーズ] 送信側が DR を送れば、いつか必ず送信側は DC を受け取り、コネクションが切断される。

(4) freq s_cr
: このプロセスは無限動作する。

4.1.3 各フェーズの詳細な性質

次に、各フェーズの詳細な性質を示す。

(1') コネクション確立フェーズ

(1'.1) (freq s_cr) ⇒ ((sometime (r_cc ⊕ s_dr))
: 送信側が何度も CR を送れば、いつかは受信側に CR が受け取られる。または、送信側が DR を送る。

(1'.2) ALWAYS (r_cr ⇒ (SOMETIME (s_cc ⊕ s_dr)))
: 受信側が CR を受け取れば、受信側は CC を送る。または送信側が DR を送る。

(1'.3) (freq s_cc) ⇒ (sometime (r_cc ⊕ s_dr))
: 受信側が何度も CC を送れば、いつかは送信側に CC が受け取られる。または、送信側が DR を送る。

(2') データ転送フェーズ

(2'.1) ALWAYS (r_cc ⇒ (SOMETIME (s_dt ⊕ s_dr)))
: 送信側が CC を受け取れば、送信側はデータを送る。または、送信側が DR を送る。

(2'.2) (freq s_dt) ⇒ (sometime (r_dt ⊕ s_dr))
: 送信側が何度もデータを送れば、いつかは受信側にデータが受け取られる。または、送信側が DR を送る。

(2'.3) ALWAYS (r_dt ⇒ (SOMETIME (s_ak ⊕ s_dr)))
: 受信側がデータを受け取れば、受信側は AK を送る。または、送信側が DR を送る。

(2'.4) (freq s_ak) ⇒ (sometime (r_ak ⊕ s_dr))
: 受信側が何度も AK を送れば、いつかは送信側に AK が受け取られるか、または送信側が DR を送る。

(2'.5) freq s_dt
: データの転送は何度でも可能である。

(3') コネクション切断フェーズ

(3'.1) ALWAYS (s_dr ⇒ (SOMETIME (r_dr ⊕ (SOMETIME r_dc))))
: 送信側が DR を送れば、受信側が DR を受け取るか、または、送信側が DC を受け取る。

(3'.2) ALWAYS (r_dr ⇒ (SOMETIME s_dc))
: 受信側が DR を受け取れば、受信側は DC を送る。

(3'.3) ALWAYS (s_dc ⇒ (SOMETIME r_dc))
: 受信側が DC を送れば、送信側は DC を受け取る。

4.2 時間的性質を満足するプロトコル仕様

詳細は省略するが、上の性質から構成法 1、 2 に従って、次のような **LOTOS** 仕様が得られる。これらの仕様自身はプロトコル全体の振舞いを示すが、[6] に従って、送信側、媒体、受信側の 3 者に分割することができる。

全体的な時間的性質からのプロトコル仕様

```
process  Proc1[s_cr, r_cc, s_dr, r_dc]:=
    s_cr; ( r_cc; s_dr; Proc2[s_cr, r_cc, s_dr, r_dc]
          [] s_dr; Proc2[s_cr, r_cc, s_dr, r_dc] )
where
    process Proc2[s_cr, r_cc, s_dr, r_dc]:=
        r_dc; Proc1[s_cr, r_cc, s_dr, r_dc]
    endproc
endproc
```

詳細な性質からのプロトコル仕様

```
process  Proc1[Δ]:=
  s_cr; Proc2[Δ]
where
  process  Proc2[Δ]:=
    hide int1 in
      int1; s_cr; Proc2[Δ]
    [] r_cr; ( s_cc; Proc3[Δ] [] s_dr; Proc7[Δ] )
    [] s_dr; Proc7[Δ]
  endproc
  process  Proc3[Δ]:=
    hide int2 in
      int2; s_cc; Proc3[Δ]
    [] r_cc; Proc4[Δ]
    [] s_dr; Proc7[Δ]
```

```
    endproc
    process  Proc4[Δ]:=
        s_dt; Proc5[Δ]
      [] s_dr; Proc7[Δ]
    endproc
    process  Proc5[Δ]:=
      hide int3 in
        int1; s_dt; Proc5[Δ]
      [] r_dt; ( s_ak; Proc6[Δ] [] s_dr; Proc7[Δ] )
      [] s_dr; Proc7[Δ]
    endproc
    process  Proc6[Δ]:=
      hide int4 in
        int4; s_ak; Proc6[Δ]
      [] r_ak; Proc4[Δ]
      [] s_dr; Proc7[Δ]
    endproc
    process  Proc7[Δ]:=
      hide int5 in
        int5; Proc8[Δ]
      [] r_dr; s_dc; Proc8[Δ]
    endproc
    process  Proc8[Δ]:=
      r_dc; Proc1[Δ]:=
    endproc
endproc
```

ただし、Δは、
s_cr, r_cr, s_cc, r_cc, s_dt, r_dt, s_ak, r_ak, s_dr, r_dr, s_dc, r_dc
である。

5 結び

本論文では、LTSから得られるアクション系列集合をモデルとする拡張分岐時制論理を定義し、時間的性質を満足するLTS仕様を合成する方法を示した。さらに、生存性等のプロトコルの時間的性質を用いて**LOTOS**仕様を導出することができることを示した。この方法では、時間的性質を段階的に与えることにより、段階的な仕様化を行なうことができる。一般に、時間的性質を満足するLTSは一通りには決定できないが、本論文ではアクション数が最小となるLTSを選んだ。また**LOTOS**の構文の内、**prefix , choice , process instantiation** のみを用いたが、今後、その他の構文も扱う予定である。

参考文献

[1] ISO/IEC, "Information Technology – Open Systems Interconnection – Guidelines for the Application of Estelle, LOTOS, SDL", TR10167, 1991.

[2] R.Gotzhein, "Temporal Logic and Applications – a Tutorial", *Computer Networks and ISDN System*, Vol.24, pp.203-218, 1992.

[3] M.Ben-Ari, Z.Manna, A.Pnueli, "The Temporal Logic of Branching Time", Proc. of 8th Annual ACM Symposium on Principles of Programming Languages, pp.164-176,1981.

[4] A.R.Cavalli, F.Horn, "Proof of Specification Properties By Using Finite State Machines and Temporal Logic", *Protocol Specification, Testing, and Verification* VII, pp.221-233, 1987.

[5] A. Fantechi, S. Gnesi, C. Laneve, " An Expressive Temporal Logic for Basic LOTOS " , *Formal Description Techniques* II, pp.261-276, 1990.

[6] 馬淵博之, 高橋薫, 白鳥則郎, " LOTOSに基づいたプロトコル仕様の導出", 電子情報通信学会, IN91-110, Sep. 1991.

[7] ISO, "Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", ISO8807, 1989.