ソフトウェア工学 89－5
（1992. 12. 14）

時制論理を用いた並列性を含むLOTOS仕様の導出

安藤敏彦†, 加藤靖†, 高橋薫††, 野口正一†††

† 仙台電波工業高等専門学校

†† 東北大学電気通信研究所

††† 東北大学応用情報学研究センター

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, 構成的な手続きを用いており, 仕様をあるクラスに制限することができる. また, 並列プロセスを記述することも可能なので, 全体のプロセスを小さなサブプロセスに分割することで, 仕様化が容易になる. また, この方法がプロトコルの仕様化にも有効であることを示す.

Derivation of LOTOS Specification Including Parallelism Using Temporal Logic

Toshihiko ANDO†, Yasushi KATO†, Kaoru TAKAHASHI††, Shoichi NOGUCHI†††

†Sendai National College of Technology
1, Kitahara, Kamiayashi, Aoba-ku, Sendai, 989-31 Japan

††Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, 980 Japan

†††Research Center for Applied Information Sciences, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, 980 Japan

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 derives 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 transition system. It is able to obtain a desirable class of specifications using this method. It also becomes easy to specify a large process because this method can derive parallel processes. We show an application of this method to a protocol as an example.

1 はじめに

分散システムの仕様を形式的に記述する目的で開発された形式記述技法 (Formal Description Technique, FDT) は, 検証, テストにおいて数学的に厳密な取り扱いが可能である. しかし, その一方, FDT の記述が各時点での振舞いを基にしているため, 局所的であり, 全体としての性質を陽に記述することができない. そのため, FDT の記述から時間的性質を導出したり, 直観的に理解することは容易ではない.

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 [1, 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. 時制論理を用いることにより, 安全性 (safety property), 生存性 (liveness property) 等の時間的性質を陽に示すことができ, 全体的な振舞いを直観的に把握しやすい. そのため, FDT で記述された仕様が時間的性質を満足しているかどうかを時制論理を用いて検証する方法が試みられている [3, 4]. また, 仕様化の際に, 希望する時間的性質を満足するような仕様を得ることができれば, 仕様化の方法として望ましい.

我々は LOTOS 記述の意味であるラベル付き遷移システム (Labelled Transition System, LTS) から導出されるアクション集合をモデルとする拡張分岐時制論理 (Extended Branching time Temporal Logic, EBTL) を定義し, EBTL で記述された時間的性質からそれを満足する LOTOS 仕様を導出する構成法を提案した [5]. この構成法に従えば, チョイス, アクションプレフィックス, プロセスインスタンシエーションを使って記述できる LOTOS 仕様を導出することができる. しかし, この構成法のみを用いて大きなプロセスの仕様を得ることは容易ではない.

そこで, 本論文では並列プロセスを記述できるよう, 時間的性質からそれを満足する並列性を考慮した LOTOS 仕様の構成法を提案する. この構成法によって, プロセスを複数のサブプロセスに分割することができ, 大きな仕様を得る事がより容易となる.

2 時制論理

2.1 EBTL

LTS から得られる分岐するアクション系列上で満足される性質を記述するため, これらのアクション系列をモデルとする EBTL を定義する.

[定義 1] Φ を原子命題全体の集合とする. EBTL の論理式を次の規則によって構成する.

(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, ALL P, some P, ALWAYS P, always P, SOMETIME P, sometime P, NEXT P, next P は論理式である.

(5) 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 .

□

定義 1 で用いられている EBTL の演算子の意味を, 一つのアクション系列をモデルとする線形時制論理 (Linear time Temporal Logic)[5] の時間演算子□ (always), ◇ (sometime), ○ (next), $\mathcal{U}$(until) $\mathcal{W}$(unless) を用いて次のように定義する.

[定義 2] EBTL の充足関係 $\models$ を, $\mathcal{M}$=⟨S, *Seq*(Sys)⟩ をモデルとし, 線形時制論理を用いて次のように定義する. ただし, S は LTS Sys の状態集合, *Seq*(Sys) は Sys のアクション系列全体の集合である. ここで, a をアクション, P, Q を EBTL の論理式, $Path^{+i}$(Sys) を Sys の状態 s_i から始まる経路全体の集合, $seq(x)$ を経路 x に対応するアクション系列とする.

$\mathcal{M}, s_i \models a$ iff (s_i, s_j)$\in$T なる s_j が唯一つ存在し, s_i から始まる経路 x に対して, $seq(x), 1 \models a$

$\mathcal{M}, s_i \models \neg P$ iff ($\mathcal{M}, s_i \models P$) ではない

$\mathcal{M}, s_i \models P \wedge Q$ iff $\mathcal{M}, s_i \models P$ かつ $\mathcal{M}, s_i \models Q$

$\mathcal{M}, s_i \models P \Rightarrow Q$ iff $\mathcal{M}, s_i \models \neg P \vee Q$

$\mathcal{M}, s_i \models P \equiv Q$ iff $\mathcal{M}, s_i \models (P \Rightarrow Q) \wedge (P \Rightarrow Q)$

$\mathcal{M}, s_i \models P \vee Q$ iff $\mathcal{M}, s_i \models \neg(\neg P \wedge \neg Q)$

$\mathcal{M}, s_i \models P \oplus Q$ iff $\mathcal{M}, s_i \models P \wedge \neg Q \vee \neg P \wedge Q$

$\mathcal{M}, s_i \models$ ALL P iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models P$

$\mathcal{M}, s_i \models$ some P iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models P$

$\mathcal{M}, s_i \models$ ALWAYS P iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Box P$

$\mathcal{M}, s_i \models$ always P iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Box P$

$\mathcal{M}, s_i \models$ SOMETIME P iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Diamond P$

$\mathcal{M}, s_i \models$ sometime P iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Diamond P$

$\mathcal{M}, s_i \models$ NEXT P iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models \bigcirc P$

$\mathcal{M}, s_i \models$ next P iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models \bigcirc P$

$\mathcal{M}, s_i \models$ P UNTIL Q iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \mathcal{U} Q$

$\mathcal{M}, s_i \models$ P until Q iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \mathcal{U} Q$

$\mathcal{M}, s_i \models$ P UNLESS Q iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \mathcal{W} Q$

$\mathcal{M}, s_i \models$ P unless Q iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \mathcal{W} Q$

□

以下では, EBTL の論理式を時間的性質 (temporal property) と呼び, LTS の時間的性質と言えば, 初期状態 s_0 での時間的性質を指すものとする. さらに, 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.

[定義 3]

$\mathcal{M}, s_i \models$ FREQ P iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Box\Diamond P$

$\mathcal{M}, s_i \models$ freq P iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models \Box\Diamond P$

$\mathcal{M}, s_i \models$ P BEFORE Q iff $\forall x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \wedge \bigcirc\Diamond Q$

$\mathcal{M}, s_i \models$ P before Q iff $\exists x \in Path^{+i}(Sys)$, $seq(x), 1 \models P \wedge \bigcirc\Diamond Q$

□

これらの演算子の内, 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.

2.2 生存性

プロセスの時間的性質で重要であるのが生存性である. 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. 例えば,

ALWAYS ((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 (確認を返す)))

: 受信側がデータを受け取れば, 受信側は送信側に確認を返す.

がそうである. これはプロセスのどの状態でも成り立たなければならない性質, つまり, 不変的 (invariant) な性質である. EBTL では生存性を陽に表現することができ, 一般に, 次のように表現することができる.

$$\text{ALWAYS}(P \Rightarrow Q). \quad (1)$$

3 時間的性質からの並列性を考慮した LOTOS 仕様の導出

この節では, 与えられた時間的性質から構成的に LOTOS 仕様を導出する方法を示す. [5] で提案した構成法では, 全体的な振舞いを示す仕様を与えたが, ここでは, 全体のプロセスを複数のサブプロセスに分割して仕様を構成する. 構成法の概要は次の通りである. (1) 与えられた各々の時間的性質を満足する LTS を求める. (2) 各サブプロセス毎にそれらの LTS を接続する. (3) 最終的に得られた各サブプロセスの LTS を LOTOS に変換し, 並列オペレータを用いて全体のプロセスを合成する.

一般に, 一つの時間的性質を満足する LTS は複数存在するが, ここでは, アクション数が最小となる LTS を選んでいる. また, 2 つの LTS を接続する時, それらが元々満足していた時間的性質が接続後も満足される保証はない. けれども, 特

定の条件の下では, 元の性質を保存するよう接続することができる. すなわち, このようにして得られたLTSは時間的性質を満足する仕様を与える上で一つの標準となる.

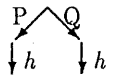
まず最初に, EBTLの演算子とLTSとの対応を示す.

[定義4] 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

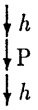
(2) P

(3) P ⊕ Q

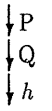


(4) ALWAYS P, always P

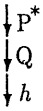
(5) SOMETIME P, sometime P



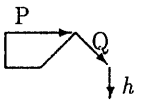
(6) P ∧ NEXT Q, P ∧ next Q



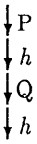
(7) P UNTIL Q, P until Q



(8) P UNLESS Q, P unless Q

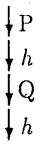


(9) P BEFORE Q, P before Q

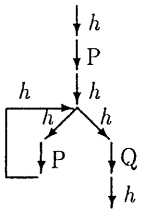


(10) FREQ P, freq P

(11) P⇒(SOMETIME Q), P⇒(sometime Q)



(12) (freq P) ⇒ (sometime Q)



□

定義4で与えたLTSは, 構成法のための部品となる. 時間的性質として生存性が与えられている時, その性質が他のLTSにおいても矛盾しなければ, 時間的性質を保存したまま, LTSをシーケンシャルに接続することができる. ここで, LTSをシーケンシャルに接続するとは, 一方のLTSの葉に, 他方のLTSの幹(根を含む部分LTS)を接続することを言う[5].

[構成法] 並列性を考慮した構成法

全体のプロセスはn個のサブプロセスからなるものとする.

(1) サブプロセス Proc_1 に対し, 満足すべき時間的性質として, 生存性を与える.

(2) 各時間的性質に対応する標準的なLTSを与える.

(3) (2)のLTSをシーケンシャルに接続する.

(4) サブプロセスの動作が有限ならば葉に正常終了アクションδを付加する. また, 動作が無限ならば, 葉を根に接続する.

(5) ループに分岐するダミーアクションhを内部アクションiに置き換える. また, 必要であれば, その他のhアクションもiに置き換えてよい. その他のhを除去する. ここで, 得られたLTSを Sys_1 とする.

(6) サブプロセス $\mathrm{Proc}_2, \ldots, \mathrm{Proc}_n$ についても(1)～(5)を行なう. ただし, 共通するアクションは全て同期しているものとして扱う.

(7) LTS $Sys_1, \ldots, Sys_n$ を [変換法] を用いて LOTOS 記述に変換する．この時，並列オペレータを使って，全体のプロセスを合成する．

□

[変換法] 並列性を考慮した LTS から LOTOS 記述への変換法

n 個の LTS $Sys_1, \ldots, Sys_n$ を LOTOS 記述に変換し，全体のプロセス Proc を導出する．

(1) **process** Proc[Δ]:=
endproc
を生成する．Δ は Procl, ..., Procn に現れるアクション全体のリストである．

(2) LTS Sys_1 に対応するプロセス名を Procl とする．Sys_1 の合流地点を各々，Procl_1, ..., Procl_p_1 とし，内部アクションを intl_1, ..., intl_q_1 とする．

(3) **process** Procl[Δ_1]:=
を生成する．Δ_1 には $Act(Sys_1)$ のアクションリストを列記する．また，根から合流地点までに内部アクションがあれば，
hide Δ_{intl} **in**
を挿入する．ここで，Δ_{intl} 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．

(4) Sys_1 の根から次の手続きを行なう．

(4.1) アクション系列が分岐しなければそのアクション系列をプレフィクス (;) を使って列記する．

(4.2) 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チョイス ([ ]) で結ぶ．

(4.3) アクション系列が葉に達したら，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，最後に
; **stop**
を付加する．また，合流地点に達したら，
; (プロセス名)
を付加する．

(4.4) 最後に
endproc
とする．

(5) Sys_1 の各サブプロセス Procl_1, ..., Procl_p_1 を Procl と同様に記述し，Procl の **endproc** の後にそれを書く．

(6) $Sys_2, \ldots, Sys_n$ のプロセス名を各々 Proc2, ..., Procn とし，これらについても，Sys_1 と同様に，(2) ～ (5) を行なう．

(7) Procl, ..., Procn の内，他のプロセスと共通するアクションを持たないプロセスを非同期並列オペレータ (|||) を使って列記し，これを $Proc_{sol1}$ とする．

(8) $Proc_{sol1}$ 以外のプロセスを番号の若い順に並べる．
$\mathrm{Proc}i_1, \mathrm{Proc}i_2, \ldots, \mathrm{Proc}i_m$
$(i_1 < i_2 < \ldots < i_m \leq n)$
そして，$\mathrm{Proc}i_2, \ldots, \mathrm{Proc}i_m$ を一つのプロセスとしてみなし，これを $Proc^1$ とする．
$\mathrm{Proc}i_1$ と $Proc^1$ に共通するアクションリストを Γ_1 とし，

$$Proc[\Delta] \equiv Proc_{sol1}[\Delta_{sol1}] \,|||\, \mathrm{Proc}i_1[\Delta_{i_1}] \;|[\Gamma_1]|\; Proc^1[\Delta^1] \qquad (2)$$

とする．ここで，Δ_{i_1}, Δ^1, Δ_{sol1} はそれぞれ $\mathrm{Proc}i_1$, $Proc^1$, $Proc_{sol1}$ のアクションリストである．

(9) $Proc^1$ を全体のプロセスとみなし，(7), (8) を行ない，

$$Proc^1[\Delta^1] \equiv Proc_{sol2}[\Delta_{sol2}] \,|||\, \mathrm{Proc}i_2[\Delta_{i_2}] \;|[\Gamma_2]|\; Proc^2[\Delta^2] \qquad (3)$$

とする．同様にして，

$$Proc^l \equiv \mathrm{Proc}i_l[\Delta_{i_l}] \qquad (4)$$

となる $\mathrm{Proc}i_l$ が現れるまで繰り返し，

$$Proc^k[\Delta^k] \equiv Proc_{solk+1}[\Delta_{solk+1}] \;|||\; \mathrm{Proc}i_{k+1}[\Delta_{i_{k+1}}] \;|[\Gamma_{k+1}]|\; Proc^{k+1}[\Delta^{k+1}] \quad (2 < k < l) \qquad (5)$$

とする．

(10) *Proc* を，式 (2) ～ (5) を用いて，Proc1, ..., Proc$n$ で表し，これを Proc の **endproc** の前に書く．

□

[注意]　この構成法では，[構成法] の (6) に明記してあるように，プロセス間に共通するアクションは全て同期しているものとしている．逆に言えば，同期していないアクションは，必ずアクション名が異なっていなければならない．□

[構成法] の例を以下に示す．

[例]　3 つのプロセス Proc_1, Proc_2, Proc_3 に次の性質を与え，それを満足するような LOTOS 仕様を導出する．

(1) 各プロセスに時間的性質を与える．

Proc_1

- ALWAYS (a⇒(SOMETIME b))
- FREQ a

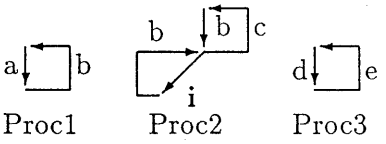
Proc_2

- ALWAYS ((freq b)⇒(sometime c))
- freq b

Proc_3

- ALWAYS (d⇒(SOMETIME e))
- FREQ d

(2) 各プロセスの LTS を構成する．LTS の接続に関する詳細は文献 [5] を参照のこと．



(3) LTS を LOTOS に変換する．この場合，Proc_1 と Proc_2 には，アクション b が共通しており，b は同期していると考える．また，Proc_3 は他のプロセスとの共通アクションを持たないので，式 (2) の $Proc_{sol_1}$ に相当する．各プロセスの関係は図 1 に示される．

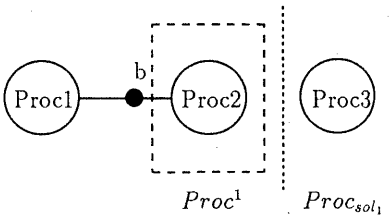


図 1　並列プロセスの例

```
process  Proc[a, b, c]:=
  Proc3[d, e] ||| Proc1[a, b] |[b]| Proc2[b, c]
where
  process  Proc1[a, b] :=
    a; b; Proc1[a, b]
  endproc
  process  Proc2[b, c] :=
    b; Proc2_1[b, c]
  endproc
  process  Proc2_1[b, c] :=
    hide int1  in
      int1; b; Proc2_1[b, c]
    [ ] c; Proc2[b, c]
  endproc
  process  Proc3[d, e] :=
    d; e; Proc3[d, e]
  endproc
endproc
```

□

4　プロトコルへの適用

本節では，この構成法をプロトコルへ適用し，LOTOS 仕様が得られることを示す．適用するプロトコルとして，アブラカダブラプロトコルのデータ転送フェーズの簡易版を扱う [6]．これは送信

側 (Sender), 受信側 (Receiver), 媒体 (Medium) の3つのプロセスから成る.

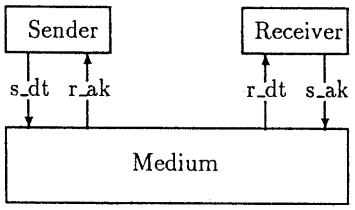


図2 データ転送フェーズの概念図

[構成法] に従って時間的性質を満足する LOTOS 仕様を導出する.

(1) 各プロセスに, 各々が満足すべき時間的性質を与える.

(1) Sender

(1.1) ALWAYS ((freq s_dt)⇒(sometime r_ak)) : データを送り続ければ, いつかその確認を受け取る.

(1.2) freq s_dt : 無限動作.

(2) Medium

(2.1) ALWAYS ((freq s_dt)⇒(sometime r_dt)) : データを送り続ければ, いつかデータが受け取られる.

(2.2) ALWAYS (r_dt ⇒(SOMETIME s_ak)) : データが受け取られれば, 確認が返される.

(2.3) ALWAYS ((freq s_ak)⇒(sometime r_ak)) ; 確認を何度も返せば, いつか受け取られる.

(2.4) freq s_dt : 無限動作.

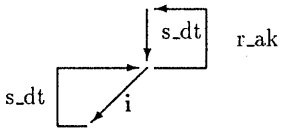
(3) Receiver

(3.1) ALWAYS (r_dt ⇒ (SOMETIME s_ak)) : データを受け取れば, 確認を返す.

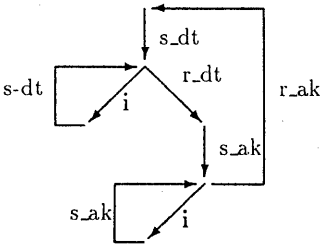
(3.2) freq r_dt : 無限動作.

(2) 各プロセス毎に各々の時間的性質を満足する LTS を導出する.

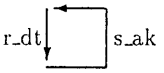
(1) Sender



(2) Medium



(3) Receiver



(3) これらの LTS を LOTOS 記述に変換する.

```
process  Proc[Δ]:=
    Sender[s_dt, r_ak]
  |[s_dt, r_ak]|
    Medium[Δ]
  |[r_dt, s_ak]|
    Receiver[r_dt, s_ak]
where
  process  Sender[s_dt, r_ak]:=
    s_dt; Proc1_1[s_dt, r_ak]
  endproc
  process  Proc1_1[s_dt, r_ak]:=
    hide int1 in
      int1; s_dt; Proc1_1[s_dt, r_ak]
    [] r_ak; Sender[s_dt, r_ak]
  endproc
  process  Medium[Δ]:=
    s_dt; Proc2_1[Δ]
  endproc
  process  Proc2_1[Δ]:=
    hide int1 in
      int1; s_dt; Proc2_1[Δ]
    [] r_dt; s_ak; Proc2_2[Δ]
  endproc
```

```
process   Proc2_2[Δ]:=
   hide int2  in
     int2; s_ak; Proc2_2[Δ]
   [] r_ak; Medium[Δ]
endproc
process   Receiver[r_dt, s_ak]:=
   r_dt; s_ak; Receiver[r_dt, s_ak]
endproc
endproc
```

これが求めるプロトコルのLOTOS仕様である．ここで，Δは

s_dt, r_dt, s_ak, r_ak

である．プロトコル全体としては，図3のように振舞う．これは，Mediumと弱バイシミュレーション関係にある．

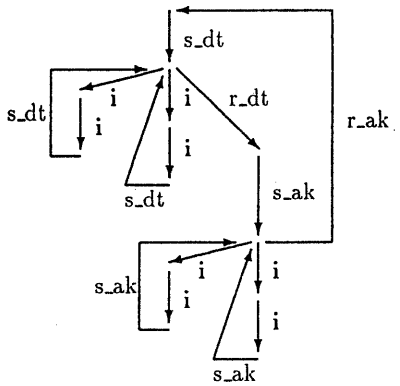


図3　プロトコル全体の振舞い

5　結び

本論文では，並列性を考慮し，EBTLで記述された時間的性質から，それを満足するLOTOS仕様を得る構成法を示した．[5]で示した構成法では，時間的性質から仕様を得ることができ，その仕様の振舞いが明確になるという利点があるが，大きなプロセスの仕様を得ることは容易ではなかった．しかし，プロセスを分割することで，より大きな仕様を構成することが可能になった．

今後の予定としては，LOTOSプロセスをアクションとみなし，そのアクション系列をモデルとする時制論理を考えている．これにより，LOTOSでの順次合成，割り込みオペレータを扱うことが可能になり，基本LOTOSの全てのオペレータを用いて仕様を構成することができる．また，この構成法の処理系の実現も予定しており，これを用いれば，仕様化がより容易になると期待される．

参考文献

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

[2] Gotzhein, R.: Temporal Logic and Applications - a Tutorial, *Computer Networks and ISDN System*, Vol.24, pp.203-218, North-Holland (1992).

[3] Cavalli, A.R. and Horn, F.: Proof of Specification Properties By Using Finite State Machines and Temporal Logic, *Protocol Specification, Testing and Verification VII*, pp.221-233, North-Holland (1987).

[4] Fantechi, A., Gnesi, S. and Laneve, C.: An Expressive Temporal Logic for Basic LOTOS, *Formal Description Techniques II*, pp.261-276, North-Holland (1990).

[5] 安藤敏彦, 加藤靖, 高橋薫, 野口正一: 時制論理に基づくプロトコルのLOTOS仕様の合成, 情報処理学会研究報告, 92-SE-87 (1992).

[6] ISO/IEC: Information Technology - Open Systems Interconnection - Guidelines for the Application of Estelle, LOTOS and SDL, ISO/IEC TR10167 (1991).

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