解　説

時相論理とその応用†

松本一教††　内平直志††
本位田真一††

1. はじめに

時間を表現するような論理体系については，古くから哲学の分野において関心を集め，時間に関する構造をどのように捉えるかといった問題を中心として議論がなされてきた．たとえば，時間を線形なものとみなしたり，分岐したものとみなしたり，あるいは環状につながったものとみなしたりする立場がある[1]．さらに，時間が離散的なものか稠密なものかという議論もある．こうしたそれぞれの認識にもとづいて，種々の異なった体系が生み出されてきた．近年の計算機科学の立場からは，とくに自然言語処理やプログラムの検証といった分野において，時間を直接表現できる体系が用いられるようになってきた．本稿では，そうした時間を表現する体系の一つである時相論理について，その応用を中心に概観する．なお，時相論理に関しては，すでにいくつかの解説[3)~5)]があるので，あわせて参考にされたい．

2. 線形時相論理と分岐時相論理

線形時相論理 (Linear Time Temporal Logic, LTL)，分岐時相論理 (Branching Time Temporal Logic, BTL) ともに，時間の進行にともなって変化していく世界の状態記述を自然に行えるように，通常の古典論理を拡張したものである．まず，LTL の一つである PTL (Propositional Temporal Logic)[6),7)] について紹介する．PTL では，次の時刻に世界がとりうる状態は唯一に決まる，という線形な時間構造にもとづいて，通常の命題論理 (Propositional Logic, PL) に時相記号 {X, F, G, U} を付加することで拡張する．PTL の論理式は，

（1） PL の論理式

（2） p, q が PTL の論理式のとき，$p \wedge q$, $\neg p$, Xp, Fp, Gp, pUq も PTL の論理式

という規則を有限回適用することで構成される．今後，$\rightarrow$, $\vee$, $\equiv$などは，通常の方法で定めておくものとする．こうして構成された論理式は，Xp (次の (世界の) 状態では p が成り立つ)，Fp (いつかは p が成り立つような状態がある)，pUq (q が成り立つ最初の状態までのすべての状態で p が成立する) などを意味する．より形式的には，構造 $K=\langle S, N, L\rangle$ に関して，論理式 p が状態 s において成立することを次のように定め，$K, s \models p$ と表す．ここに，S, N, L は次のようなものである．

S は空でない集合 (要素を状態という)

$N: S \rightarrow S$, どの状態に対してもその次の状態を与える後継者関数，N^i でこの N 回の適用を表す．

$L: S \rightarrow 2^{Prop}$, その状態で成立する命題変数を表す．(Prop は，命題変数の集合)

$K, s \models p$ (命題変数のとき) iff $p \in L(s)$

$K, s \models \neg p$ iff $K, s \models p$ でない

$K, s \models p \wedge q$ iff $K, s \models p$ かつ $K, s \models q$

$K, s \models Xp$ iff $K, N(s) \models p$

$K, s \models Gp$ iff 任意の $0 \leqq i$ に対し，
$K, N^i(s) \models p$

$K, s \models Fp$ iff ある $0 \leqq i$ に対し，$K, N^i(s) \models p$

$K, s \models pUq$ iff ある $0 \leqq i$ に対し，
$K, N^i(s) \models p$ かつ任意の $0 \leqq j < i$ に対し，
$K, N^j(s) \models q$

PTL の解釈とは，構造 K と K の状態 s_0 の組 $\langle K, s_0\rangle$ のことである．s_0 を初期状態という．解釈 $I=\langle K, s_0\rangle$ が論理式 p を満たすとは，$K, s_0 \models p$ が成立することであり，このときの解釈を p の Kripke モデルという．

例　$G(p \rightarrow Xq) \wedge G(q \rightarrow Xp)$ に対し，次の $\langle K, s_0\rangle$, $K = \langle S, N, L\rangle$ はそのモデルである．

$S = \{s_0, s_1\}$, $N(s_0) = s_1$, $N(s_1) = s_0$,

$L(s_0) = \{p\}$, $L(s_1) = \{q\}$

これは図-1 のような有限グラフで示される世界列を

† Temporal Logics and Their Applications by Kazunori MATSUMOTO, Naoshi UCHIHIRA and Shinichi HONIDEN (Systems and Software Engineering Laboratory, TOSHIBA corporation).

†† (株)東芝システム・ソフトウェア技術研究所

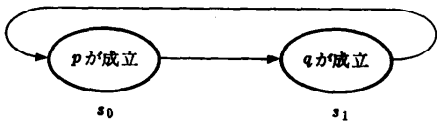

図-1 モデルの有限グラフ表現

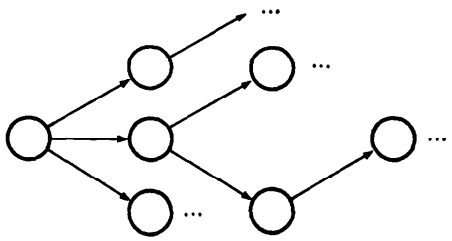

図-2 分枝時相論理の構造

考えていることに対応する．実は，充足可能な論理式は，必ずこのような有限グラフで表現できる[8]（有限モデル性）．

BTL は，図-2 のように次の時刻において世界が取りうる状態に複数の可能性があるものと考えて，PL を拡張する．このような枝分かれした状況をどのように表現するかに応じて，種々の提案があるが[9)~11)]，ここでは，Emerson[10),12)] らによって提案された CTL (Computation Tree Logic) およびその拡張である CTL* について紹介する．まず CTL* では，記号 {A, E} を，先の PTL に追加する．その論理式は，

（3） PL の論理式

（4） 疑似論理式 p に対し，Ap, Ep は CTL* の論理式

（5） CTL* の論理式 p, q に対し，$\sim p$, $p \wedge q$ は CTL* の論理式

という規則から構成される．ただし，疑似論理式とは，次の(5)と(6)で約束される．

（6） CTL* の論理式

（7） 疑似論理式 p, q に対し，$p \wedge q$, $\sim p$, Xp, Fp, Gp, pUq は疑似論理式

Ap によって，どのパス上の状態においても，p が成立するということを表し，Ep によって，あるパス上の状態では p が成立するということを表す．たとえば，

$$A(G\sim p \vee Fp) \qquad (1.1)$$

により，“どの枝においても，その枝上のすべての状態で $\sim p$ が成立しているか，または枝上のどこかに p が成立する状態がある”，ということを表す．

CTL* の形式的な意味は，PTL と同様の構造 $K=\langle S, N, L\rangle$ を用いて，定められる．ただし，分岐した構造を表現するため，N により定まる後継者状態は唯一である必要はない．

CTL は，A および E の直後には必ず {X, F, G, U} だけを許すという構文上の制限を CTL* に加えたものである．すなわち，CTL の論理式は，

（8） PL の論理式

（9） CTL の論理式 p, q に対して，$\sim p$, $p \wedge q$, AFp, AGp, AXp, $A(pUq)$, EFp, EGp, EXp, $E(pUq)$ は CTL の論理式

なる規則から構成される．たとえば，前述の (1.1) は CTL* の論理式であるが，CTL の論理式ではない．表現力に関しては，明らかに CTL は CTL* に包含されるが，4. で示すように CTL には応用上の有益な特長が見いだされている[12),13)]．

3. プログラムの自動生成への応用

形式的な仕様記述から自動的にプログラムを生成することに関しては多数の研究がある．時相論理の論理式を仕様とみなすことによる，プログラム自動生成については，Wolper[6)] や Clarke[14)] らの研究がある．

Clarke[14)] の方法は，並列プログラムの synchronization skelton を自動生成するというものであった．Synchronization skelton とは，並列プログラムの同期に関する制御を行うもので，同期に関して本質的でない部分を無視したものである．たとえば，相互排除 (mutual exclusion) 問題では，プロセスがきわどい領域 (critical section) にあるか，それ以外の領域にあるかが重要な問題であるから，各領域内部で行われる実際の処理は無視する．以降，m 個の並列プロセスの相互排除問題を考える．なお，並列実行を，各プロセスの原子動作 (atomic action) のインタリービング (interleaving) として扱う．すなわち，原子動作 p と q を並列実行することを，p, q あるいは q, p のいずれかの順に逐次実行してもよいこととみなす．m 個の並列プロセスの実行の過程は，インタリービングの選び方により分岐が生じる計算木として表現できる．この計算木が満足すべき性質を，仕様として CTL で記述する．いま，$m=2$ として，各プロセス i ($i=1, 2$) は，きわどい領域でない領域 (NCS_i)，きわどい領域に入ろうとする領域 (TRY_i)，きわどい領域 (CS_i) の三つの領域を NCS_i, TRY_i, CS_i, NCS_i, TRY_i, CS_i, NCS_i, … のように遷移するものとする．すると，

$$NCS_1 \wedge NCS_2 \qquad (3.1)$$

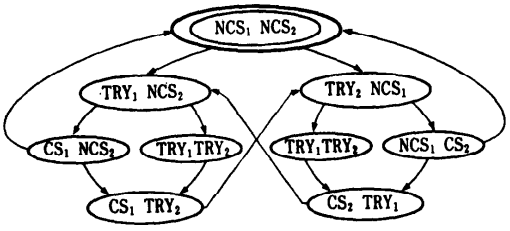

図-3 仕様のモデル（文献[14]，p. 67 より）

$$AG(\sim(CS_1 \wedge CS_2)) \tag{3.2}$$

$$AG(TRY_1 \rightarrow AFCS_1) \wedge AG(TRY_2 \rightarrow AFCS_2) \tag{3.3}$$

を論理積 ^ で結んだものとして，仕様を記述できる．ここで，(3.1) は各プロセスともきわどい領域でない部分から出発することを，(3.2) はともにきわどい領域に入ることはないことを，(3.3) は飢餓 (starvation) に陥らないことを表す．この仕様に対して，CTL の有限モデル性[8] を利用したタブロー法[9),14)]という決定手続きを適用する．タブロー法では，充足可能な論理式に対し，その有限状態のモデルが実際に構成される．図-3 は，上の仕様に対してタブロー法を適用した結果の一部を，グラフ表現したものである．

このグラフにおいて，ノードの集合を S，エッジの集合を N，ノード s に書かれた論理式を $L(s)$ の値とみなして得られる構造 $K=\langle S, N, L\rangle$ と，二重線で示されたノード s_0 に関して，$\langle K, s_0\rangle$ は仕様のモデルとなる．したがって，s_0 から出発して，グラフ上をエッジに従って遷移し，各ノードの論理式がその時点において成立するように各プロセスを動作させれば，仕様が満足されることになる．共有メモリの存在を仮定すれば，そのように各プロセスを動作させることは困難ではない．

Wolper ら[6] は PTL を用いた，上と同様な方法を提案している．すなわち，PTL の論理式による仕様記述からタブロー法により，そのモデルを構成するわけである．両者を比較するとき，PTL と CTL の，

（1） 仕様記述言語としてみた場合の表現力

（2） 論理式からモデルを生成するための計算量の違いが主要な問題となる．（1）については，それぞれに応じた特長があるため，議論の対象とされてきている．このことに関しては，Wolper[6] p. 19, Lamport[11] や Emerson[10] を参考にされたい．とくに，文献[10] においては種々の LTL, BTL の表現力に関する階層性が明らかにされている．（2）については，文献[15] を参考にされたい．

4. プログラムの検証への応用

プログラムの検証は，計算機科学における重要な研究テーマである．時相論理を用いての検証については，逐次プログラムに関する Burstall[16] の研究や並列プログラムに関する Pnueri[17] の研究を始めとして，活発な研究がある[7),18)]．本誌[4] においても，PTL を用いた場合の検証について解説がなされているので参考にされたい．時相論理を用いるプログラム検証の多くは，プログラムの性質を時相論理を用いて記述しておき，そこから検証すべき論理式を演繹的に導くというものであり，検証の自動化の問題や計算量の問題が残されていた．一方，Clarke[12),13)] や Emerson[19] らは，モデルチェッカとよばれる検証の方法を提案している．それは，プログラムの多くが，たとえば図-3 のように，有限状態遷移図 M とその初期状態 s_0 を与えることによって表現できることと，CTL の有限モデル性に着目し，その M が CTL の論理式として記述された性質を満足するかどうか検証する．すなわち，3. で行ったように，ノード，エッジ，ノードのラベルから定まる構造 $M=\langle S, N, L\rangle$ と s_0 に対して，$\langle M, s_0\rangle$ が検証すべき論理式のモデルになっているかどうかの判定である．ここでは，文献[12] に従って，モデルチェッカの概要について紹介する．

CTL によるモデルチェッカの利点は，きわめて高速な検証が行えるアルゴリズムが，見いだされていることにある．そのアルゴリズムは，与えられた論理式 p と状態遷移図 M に対し，$|p|$ (p に現れる記号の総数) ステップで終了するもので，第 i ステップにおいては，p の長さ i の部分論理式の処理を行う．各ステップにおいて，遷移図中のノードのラベルと部分論理式の照合あるいは新たなラベル付けを行い，第 i ステップ終了後に，各状態 s には，$M, s \vDash q$ を満たすすべての，長さ i 以下の論理式 q がラベル付けされる．いま，部分論理式が EGp の形をしている場合を考えよう．この処理には，次の補題を利用する．

補題 (Clarke[13], p. 274)

遷移図 M を，$M, s' \vDash p$ を満たす状態 s' だけに制限したものを M' とする．このとき，$M, s \vDash EGp$ が成立することと，次の（1），（2）がともに成立することは同値である．

（1） s は M' に含まれる状態である．

（2） M' のある強連結成分 C に対し，s から C のいずれかのノード t へ至るパスが存在する．

この補題により，$M,s\vdash EGp$ の判定には，まず遷移図 M を強連結成分に分割し，そこに属するノードを逆に辿って到達できるノードが，M' に属しているかどうかを判定すればよい．p に対する処理は，これ以前のステップで終了しているから，M' のノードかどうかの判定はノードのラベルを比較するだけでよい．こうして，あるノードで EGp が成立することが分かれば，そのノードに EGp をラベル付けし，EGp の処理を終了する．結局，EGp の処理には，強連結成分を見いだす時間と，ノードを辿りラベルの判定をする時間が必要なので，$O(|M|+|R|)$ の計算量を要することになる．ただし，$|M|$ は遷移図の状態数，$|R|$ はエッジの総数である．他の形も，これ以下の計算量で判定できるため[12]，アルゴリズム全体で，$O(|p|(|M|+|R|))$ の計算量しか必要としないことになる．このような高速性のために CTL のモデルチェッカは，プログラムの検証だけではなく，論理回路の検証[20),21)]など，状態数がきわめて大きくなる場合にも適用され，実用時間での検証が終了したと報告されている．

図-3 の相互排除の問題を考えよう．この遷移図において，(NCS_1, NCS_2), (TRY_1, NCS_2), (CS_1, NCS_2), (NCS_1, NCS_2), (TRY_1, NCS_2),… を繰り返すような遷移を行った場合，プロセス 2 は決してきわどい領域に入れないことになる．これは公平性が満たされない場合である．実際のプログラム検証では，公平なスケジューラを仮定して，公平性が保たれるような遷移だけに限定した検証を行いたい場合がある．ところが CTL では，公平性に関する性質を表現することはできないことが，知られている[10]．一つの解決方法は，公平性の記述ができる CTL* を用いることである．しかしながら，CTL* のモデルチェッカには，計算量の問題がある (PSPACE-complete)[12]．そこで，Clark ら[12] は CTL* を用いるのではなく，CTL のモデルチェッカを拡張することにより，公平性をこめた検証を行う方法を提案した．すなわち，状態遷移図以外に命題変数の集合 F を与え，その各命題変数が無限回出現するような遷移のみを対象とするよう，前記のモデルチェッカを改良する．たとえば，図-3 の遷移図の検証では，プロセス 1，2 の公平実行を仮定するため，$F=\{CS_1, CS_2\}$ を与える．具体的アルゴリズムは，文献[12]を参考にされたいが，この方法では CTL に比べさほど計算量が増加しない ($O(|p||F|(|M|+|R|))$) という利点がある．しかし，この方法では強公平性 (strong fairness) などを扱うことはできないため，次の FCTL が考案されている．

Emerson[19] らは，強公平性などの検証が可能となるように，FCTL (Fair CTL) とよばれる体系を提案した．この体系では，PTL と CTL を組み合わせた形で用いる．すなわち，FCTL では，CTL 論理式 p と PTL 論理式 ϕ との組 (p,ϕ) により，検証すべき性質を記述する．ただし，ϕ の時相記号は F と G だけに制限しておく．直観的には，状態遷移図 M を一つの枝に制限したものは，PTL の構造とみなせることに着目し，M を ϕ が成立する枝に限定した遷移図 M' に対して，CTL の論理式 p が成立するかどうかを検証する．PTL では CTL では記述できない，種種の公平性に関する記述ができるため[10]，このように PTL と CTL を組み合わせることが有効となる．

FCTL で記述された仕様 (p,ϕ) に対する意味は，CTL* の論理式への変換によって与えられる．たとえば，(AFp, Fq) は，$A(Fq\rightarrow AFp)$ なる CTL* 論理式を意味する．ϕ が $\bigwedge_{i=1,n}\bigvee_{j=1,m}(G^{\omega}p_{ij}\vee F^{\omega}q_{ij})$ なる形式 (標準形) で記述された場合には，$O(|M||p||\phi|^2)$ で終了するモデルチェックのアルゴリズムが提示されている ($F^{\omega}p$ は GFp を，$G^{\omega}p$ は $\neg F\neg p$ を表す)．標準形以外で ϕ が表現された場合には，標準形への変換のために $|\phi|$ の指数関数的な時間が必要になるが，種々の公平性に関しては，

1. (無条件) 公平性
$$\bigwedge_{i=1,n} F^{\omega}\ \text{executed}_i$$
2. 弱公平性
$$\bigwedge_{i=1,n}(G^{\omega}\ \text{enabled}_i\rightarrow F^{\omega}\ \text{executed}_i)$$
$$=\bigwedge_{i=1,n}(F^{\omega}(\neg\text{enabled}_i\vee\text{executed}_i))$$
3. 強公平性
$$\bigwedge_{i=1,n}(F^{\omega}\ \text{enabled}_i\rightarrow F^{\omega}\ \text{executed}_i)$$
$$=\bigwedge_{i=1,n}(G^{\omega}\neg\ \text{enabled}\vee F^{\omega}\ \text{executed})$$

のように標準形で記述できるため，公平性に関する検証に際しては変換の問題は生じない．

PTL に対しても，モデルチェッカを考えることができる[22),23)]．この場合，遷移図の状態数に関しては線形であるが，論理式の長さに関して指数関数的な計算量が必要となる．このことに関しては，現実的には遷移図の状態数のほうが論理式の長さより問題となる，という主張もあり[23)]，PTL か CTL かについての論争がある[19),23)] (文献[19)]のタイトルに注意されたい)．

5. 時相論理の拡張

応用的な観点からは，より表現力があり自然な記述ができる体系を用いることが望ましい．実際の適用分野に応じて，時相論理の拡張が試みられている．

多ソート時相論理 (many-sorted temporal logic) は，Enjalbert[24] らによって提案された．これまでに紹介してきた PTL や CTL では，複数の，各々の時間にもとづいて動くプロセスからなる並列システム全体を記述することは困難であった．多ソート時相論理は PTL に，ソートの概念を導入したもので，プロセスをソートに対応させることで，そのような並列システムの記述を容易に行える．

多ソート時相論理は，$1,\cdots,k$ および ε で示される $k+1$ 個のソートからなる．これらの各ソート i ($i=1,\cdots,k$) に対し，PTL と同様に時相記号 $\{X^i, G^i, F^i\}$ を用いて，その論理式が定められる．ソート i 上の論理式を p^i，その集合を Wff^i と表す．ソート ε は，ある時刻におけるシステム全体のようすを表現するためのものであり，その論理式は次を満たす最小の集合 Wff の要素として約束される．

(1) 各 i に対し，$Wff_i \subseteq Wff$

(2) $p, q \in Wff$ ならば $p \wedge q, \neg p, Xp, Gp \in Wff$

こうして定められた論理式の意味は，$k+1$ 組の構造 $K=\langle K^1,\cdots,K^k,K'\rangle$ を用いて与えられる．ここで各 K^i は，PTL と同じ構造であり，状態の集合 S^i，関係 N^i および L^i の三つ組である．各ソートの状態を組にした $s=\langle s^1, s^2,\cdots, s^k\rangle$ を大域状態とよぶ．K' は大域状態を図-4 のように並べたものである．

ここで $s_j{}^i$ は，$N^i(s_j{}^i)=s^i{}_{j+1}$ または $s_j{}^i=s^i{}_{j+1}$ を満たすソート i の状態である．こうした上で各ソート i において，論理式 p^i が状態 s^i において成立することを PTL と同様に定める．ソート ε 上の論理式 p が成立するということを大域状態を用いてこれまで

$$\begin{array}{l}\langle s_1{}^1, s_1{}^2, \cdots, s_1{}^k\rangle\\ \langle s_2{}^1, s_2{}^2, \cdots, s_2{}^k\rangle\\ \quad\vdots\\ \langle s_i{}^1, s_i{}^2, \cdots, s_i{}^k\rangle\\ \quad\vdots\end{array}$$

図-4 大域状態の列

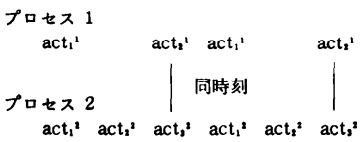

図-5 二つのプロセスからなるシステム

と同様に定めることができる．

この体系による簡単なシステムの記述を行ってみよう．二つのプロセスからなるシステムを考える．プロセス 1 は，$act_1{}^1$，$act_2{}^1$ を交互に実行し，プロセス 2 は，$act_1{}^2$, $act_2{}^2$, $act_3{}^2$ を順に繰り返す．さらに，$act_2{}^1$ と $act_3{}^2$ は同時に実行されるものとする (図-5)．このような要求は，プロセス 1 について，

$act_1{}^1$,

$G^1(act_1{}^1 \rightarrow X^1\ act_2{}^1)$, $G^1(act_2{}^1 \rightarrow X^1\ act_1{}^1)$

$G^{1\sim}(act_1{}^1 \wedge act_2{}^1)$

プロセス 2 について，

$act_1{}^2$,

$G^2(act_1{}^2 \rightarrow X^2\ act_2{}^2)$, $G^2(act_2{}^2 \rightarrow X^2\ act_3{}^2)$,

$G^2(act_3{}^2 \rightarrow X^2\ act_1{}^2)$,

$G^{2\sim}(act_1{}^2 \wedge act_2{}^2)$, $G^{2\sim}(act_2{}^2 \wedge act_3{}^2)$,

$G^{2\sim}(act_1{}^2 \wedge act_3{}^2)$

さらに，同時実行の条件を

$G(act_2{}^1 \leftrightarrow act_3{}^2)$

のようにして記述できる．

マルチプロセッサ上で実行されるプログラムの考察には，時間の概念に加えて，プロセッサの位置関係という空間の概念が重要になる場合がある．そこで Reif ら[25] は，時相論理に空間を表す記号 {somewhere, everywhere} を追加した体系 multiprocessor network logic を提案した．これを用いれば，

G everywhere (sent→somewhere GF received)

によって，どの場所でどの時間に送信されたメッセージもどこかの場所でいつかは必ず受信される，のような表現を行うことができる．

6. 時相論理の表現力

前章までに，いくつかの時相論理について紹介してきたが，それらは命題論理にもとづくものであった．実際の応用においては，述語論理の拡張である時相述語論理が有効な場合がある．たとえば，無限個のデータすべてに関してある性質が成立するという要求は，PTL (CTL) では記述できない．反面，時相述語論理を用いた場合，3., 4. のような簡潔な生成，検証を行えないという問題点がある．このような問題に対し，Wolper[26] は，記述対象にある制限を仮定すれば，PTL (CTL) による無限のデータ間の関係が記述可能になるという，興味深い結果を導いた．文献[26]に従って，このことについて紹介する．

いま，可算集合 D を考える．D をデータ領域，D

の要素をデータとよぶ．データを読み込み，それを変更なしに出力することを，並列的に，無限に続けるようなプログラムを simple reactive プログラムとよび，以降，そのようなプログラムに限って議論する．

3. と同様に，データ領域 D に対して動くプログラムを，インタリービングにより，$e=e_1, e_2, e_3, \cdots$ で表す．e をプログラムの動作列とよぶ．ただし，各 e_i は $\mathrm{in?}d_k$（データ d_k を読み込んだ）または，$\mathrm{out!}d_m$（データ d_m を出力した）を表す命題変数である．

さて，すべて異なったデータからなる無限列 $\sigma \in D^\omega$ を読み込みながら，σ の要素間の順序を変えずに出力していくプログラムを考える．このとき，

要求(1) d_i が d_j より先に読み込まれたものであれば，d_i を d_j より先に出力する

要求(2) プログラムが，データ読み込みを停止することはない

などを記述する必要がある．いまこれらの要求が，σ の要素間の順序だけを問題にしており，D については言及していないことに注目してみると，次のデータ独立という性質が成立している．

定義 (Wolper[26])

プログラム P がデータ独立であるとは，データ領域 D と任意の関数 $f: D \to D'$ に対して，e が P の仕様を満たす動作列であることと，$f(e)$ が P の仕様を満たす動作列であることとが同値であることである．($f(e)$ は，e 中の $\{\mathrm{in?}d_1, \mathrm{out!}d_1, \cdots\}$ を $\{\mathrm{in?f}(d_1), \mathrm{out!f}(d_1), \cdots\}$ で置き換えたものである)

いま，領域 D に関するプログラム P の仕様を $\pi(P, D)$ と書くことにすれば，次の命題が成り立つ．

命題 (Wolper[26])

データ独立なプログラム P と，任意の全射関数 $f: D \to D'$ に対して，$\pi(P, D')$ の成立と $\pi(P, f^{-1}(D'))$ の成立は同値である．

いま考察中のプログラムは，データ独立であるから，先の要求(2)は，この命題により，$f: D \to \{d'\}$ を考えて，次の PTL 論理式で表現できることになる．

$$\mathrm{GF\ in?}d'$$

さらに，次の結果が成立する，

定理 (Wolper[26])

データ独立なプログラム P において，データ領域 D とその有限部分集合 D_0, D_0 を1対1に $f(D_0)$ に写すような $f: D \to D'$ を考えると，$\pi(P, D_0)$ の成立と $\pi(P, f(D_0))$ の成立は同値である．

この定理より，先の要求(1)に対して，特定の二つのデータ $\{d_1, d_2\}$ に対して，$\pi(P, \{d_1, d_2\})$ が成立すれば，任意の二つのデータに対して $\pi(P, \{d_i, d_j\})$ が成立することが分かる．結局，これと先の命題を併せて，

$$\mathrm{G}(\neg\mathrm{in?}d_1 \cup \mathrm{in?}d_2) \to \mathrm{G}(\neg\mathrm{out!}d_1 \cup \mathrm{out!}d_2)$$

として要求(1)を表現できた．

3., 4. では，固定された m 個のプロセスの相互排除問題を考察した．それは具体的に m が与えられれば，生成や検証が行えるものであった．この Wolper の研究は，そうした方法を任意の m について適用できるよう，つまり m をパラメータとしてもつようなプログラムを生成あるいは検証できないか，という動機にもとづいている．これは，データ独立という制限のない一般の場合には不可能であることがすでに証明されている[27]．この方法は，プロトコルや種々のバッファの検証などに応用できることが指摘されている[26]．

7. おわりに

本稿では，時相論理とその応用について紹介した．ここで紹介した以外にも，岩沼ら[28),29)]による multi processor network logic をさらに拡張した体系の提案，Hart ら[30)]による確率時相論理 (Probabilistic temporal logic) の提案などがある．また，モデルチェッカを大規模な回路検証などに適用した場合，状態遷移図が巨大なものになってしまうという問題がある．この解決のために，階層的な検証を行う方法が提案されている[31)]．

参考文献

1) Rescher, N. and Urquhart, A.: Temporal Logic, Springer-Verlag (1971).
2) Galton, A. (ed.): Temporal Logics and Their Applications, Academic Press (1987).
3) 堂下，西田，三浦：様相論理とその情報処理への応用(I)，情報処理，Vol. 29, No. 1 (1988).
4) 堂下，西田，三浦：様相論理とその情報処理への応用(II)，情報処理，Vol. 29, No. 2 (1988).
5) 原尾，岩沼：プログラム理論と様相論理，数理科学，No. 5 (1986).
6) Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications, Ph. D thesis of Stanford Univ. (1982).
7) Kröger, F.: Temporal Logic of Programs, EATCS Monographs on Theoretical Computer Science, Vol. 8, Springer-Verlag (1987).
8) Emerson, E. and Sistila, A.: Deciding Full

Branching Time Logic, Inf. Control, Vol. 61, No. 3 (1984).
9) Ben-Ari, M., Pnueli, A. and Mann, Z.: The Temporal Logic of Branching Time, Acta Inform., Vol. 20 (1983).
10) Emerson, E. and Halpern, J.: "Sometimes" and "Not Never" Revisited: on Branching Versus Linear Time, J. ACM, Vol. 33, No. 1 (1986).
11) Lamport, L.: "Sometimes" is Sometimes "Not Never", 7th ACM POPL (1980).
12) Clarke, E., Emerson, E. and Sistila, A.: Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications, ACM TOPLAS, Vol. 8, No. 2 (1986).
13) Clarke, E. A. and Grümberg, O.: Research on Automatic Verification of Finite State Concurrent Systems, Ann. Rev. Comput. Sci. (1987).
14) Clarke, M. and Emerson, A.: Design and Synthesis of Synchronization Skeltons Using Branching Time Temporal Logic, LNCS Vol. 131.
15) Sistiala, A. and Clarke, E.: The Complexity of Propositional Linear Time Temporal Logic, J. ACM, Vol. 32, No. 3 (1985).
16) Burstall, M.: Program Proving as Hand Simulation with a Little Induction, Proc. IFIP Congr. 1974, North Holland (1974).
17) Pnueli, A.: The Temporal Semantics of Concurrent Programs, 18th Symp. Found. Comput. Sci. (1977).
18) Manna, Z. and Pnueli, A.: Verification of Concurrent Programs: The Temporal Frame Work, in The Correctness Problem in Computer Science, Academic Press (1981).
19) Emerson, E. A. and Lei, C.: Modalities for Model Checking: Branching Time Logic Strikes Back, Science of Computer Programming (1987).
20) Browne, H., Clarke, E. and Dill, D.: Automatic Verification of Sequential Circuits Using Temporal Logic, IEEE Trans. on Comput., Vol. 33, No. 12 (1986).
21) Dill, D. and Clarke, E.: Automatic Verification of Asynchronous Circuits Using Temporal Logic, IEEE Proc. Vol. 133, Pt. E, No. 5 (1986).
22) Vardi, M. and Wolper, P.: An Automata-theoretic Approach to Automatic Program Verification, Proc. Conf. Logic in Comput. (1986).
23) Lichtenstein, O. and Pnueli, A.: Checking That Finite State Concurrent Programs Satisfy Their Linear Specification, 12th ACM POPL (1985).
24) Enjalbert, P.: Many-Sorted Temporal Logic for Multi-process Systems, LNCS 176 (1984).
25) Reif, J. and Sistila, A. P.: A Multiprocess-network Logic with Temporal and Spatial Modalities, JCSS, Vol. 30, No. 1 (1985).
26) Wolper, P.: Expressing Interesting Properties of Program in Propositional Temporal Logic, 13th ACM POPL (1986).
27) Apt, K. R. and Kozen, D. C.: Limits for Automatic Verification of Finite-State Concurrent Systems, Inf. Processing Letters (1986).
28) 岩沼，原尾：時間と空間を扱う様相述語論理の不完全性とその相対的完全化，信学論，Vol. 70 D, No. 5 (1987).
29) 岩沼，原尾，野口：時空間様相論理 ETSL の完全無矛盾な公理系，信学論，Vol. 69 D, No. 4 (1986).
30) Hart, S. and Sharir, M.: Probabilistic Temporal Logics for Finite and Bounded Models, Proc. 16th STOC (1984).
31) Mishra, B. and Clarke, E. M.: Hierarchical Verification of Asynchronous Circuits Design Using Temporal Logic, Theor. Comput. Sci. (1985).

(平成元年3月29日受付)