知識工学と人工知能 60 － 3
（１９８８．９．２１）

時制論理に基づく実用的仕様記述言語ＰＴＳからの手順の自動生成

内平直志，川田秀司

株式会社東芝　システム・ソフトウェア技術研究所

時制命題論理（ＰＴＬ：Propositional Temporal Logic）を基礎とし，対象の状態と対象への作用の記述を可能とした仕様記述言語ＰＴＳ（Practical Temporal Specification Language）を提案する．ＰＴＳで記述した対象の状態と作用についての仕様から，仕様を満す全ての手順を自動生成することができる．ＰＴＳはＰＴＬの記述能力と同等であるが，記述の容易さ，生成の効率ともにＰＴＬより優っている．また，システムが複雑になると手順の組み合せが爆発的に多くなり，手順生成には膨大な計算が必要になる．ところが，実際はユーザにとって局所的な手順は比較的自明であることが多い．そこで，ＰＴＳでは局所手順記述を導入し，ユーザが局所的な手順を有限オートマトンとして記述できるようにした．これにより，手順の組み合わせ的爆発をある程度緩和できる．最後に，ＰＴＳを電力系統の系統切替え手順の生成に適用し有効性を確認する．

PRACTICAL TEMPORAL SPECIFICATION LANGUAGE AND PLAN SYNTHESIS

Naoshi Uchihira,　　Hideji Kawata

Systems & Software Engineering Lab. TOSHIBA Corporation
Yanagicho 70, Saiwai-ku, Kawasaki 210, JAPAN
PHONE:(044)548-5465

The Practical Temporal Specification langage (PTS) is proposed, which is based on propositional temporal logic (PTL). PTS is designed for plan synthesis. We can synthesize all possible plans from a specification written in PTS. PTS has same representative ability as PTL, but PTS is easier to write and quicker to synthesize. It is reason why it is called "practical language". Another unique feature of PTS is the local plan description. In PTS specification, we can describe local plans in the finite automata form. The local plans are helpful in avoiding combinatorial explosion. We show an example of plan synthesis in the power switching system using PTS.

1. はじめに

時間関係を陽に扱う論理で記述された仕様から，その仕様を満す具体的解（モデル）を生成する研究は，ロボットの計画生成やプログラム合成の分野で行われてきた．特に，様相論理の一つである時制命題論理（ＰＴＬ）は決定可能であるため，自動生成が比較的容易であり，並列プログラムの同期部の自動生成にＰＴＬを用いたMann a&Wolperの研究［MW84］をはじめとしてプログラム合成手法やロボットの手順生成手法［Stu85 ］が提案されてきた．プログラムは計算機の実行する手順の列であるから，プログラム生成も広い意味での手順生成と考えることができる．

ＰＴＬで扱う対象は有限状態システムの手順生成であり，すべての場合を尽すことが可能であるから，本質的に手順生成可能である．しかし，組合せ的爆発が問題となる．実用化のためには効率的生成手法が不可欠である．また，対象および目的が有限状態システムの手順生成に特定化されている場合，ＰＴＬは原理的に記述可能ではあっても，かならずしも記述が容易であるとはいえない．すなわち，対象の状態と対象を変化させる作用という２つの概念を陽に区別して扱えない．さらに，状態は作用が為されないかぎり保持されるのが自然である．この状態の保持もＰＴＬでは１つ１つ記述する必要がある（フレーム問題）．このような，状態（state ）と作用（action）および作用による状態の変化（change）に関する問題はプランニングの分野で古くから研究されており，文献［GN87］にまとめられているが，時制論理によるアプローチはあまりない［Fusa83］．作用が起こらなければ，“デフォルト”として状態が保持されるという観点から，非単調論理をＰＴＬに組み込んだアプローチ［佐伯］もあるが，決定手続きの計算量は増大し，実用的手順生成には適さない．

本論文では，ＰＴＬに対象の状態と対象への作用を陽に記述できるようにした手順生成のための実用的仕様記述言語ＰＴＳ（Practical Temporal Specification language ）を提案する［川田87］．ＰＴＳにおいて，手順の基本単位は，対象の状態を知り対象の状態を変えることであり，この単位をメソッドと呼ぶ．メソッドの列が手順である．そして，ＰＴＳで記述された仕様を充足する全ての手順を生成する手法を述べる．ＰＴＳでは，変化をおこす作用はメソッドに限定されてるので，手順生成手続きも汎用的なＰＴＬの決定手続きより特殊化でき高速化できる．さらにＰＴＳでは，局所的な手順があらかじめわかっている場合に，それを事前情報として取り込み，より効率的な手順生成が可能である．

時間を扱う論理には，もう１つの別の流れがある．それはAllen の時区間論理［Allen84］やKowalskiのEvent Calculus［Kowa86］である．前述のＰＴＬやＰＴＳは様相論理に基づいているが，Allen の時区間論理やEvent Calculusは，一階述語論理（Prolog）に，時間に関する表現と推論規則を付加した論理である．時区間論理でもＰＴＳ同様に手順生成や順序の検証が可能である［西村87，丸田87］．時区間論理とＰＴＳとの相違は，ＰＴＳが手順の無限系列を扱えるのに対して，時区間論理は有限系列しか扱えないこと，時区間論理は論理的正当性が保証されないが記述力が高いなどの点である．無限に動く並行プログラムの自動生成や検証などにはＰＴＬやＰＴＳが適している[Pnue81,Clar86]．

本論文では，２，３，４章でＰＴＳの言語仕様，ＰＴＬとの関係を説明し，５章で手順の自動生成手法を述べる．６章で，ＰＴＳの効率をＰＴＬとの比較によって評価し，７章でＰＴＳを電力系統操作手順生成に適用した例を述べる．

2．ＰＴＳの動機と基本概念

2．1　ＰＴＬの問題点

線形時制命題論理（ＰＴＬ）は，文献［MW84］に基づいて次のように定義する．

（定義）

Ｐを原子命題の集合とするとき，

（１）全ての原子命題ｐ∈ＰはＰＴＬ論理式．

（２）ｆ，ｆ１，ｆ２がＰＴＬ論理式ならば，¬ｆ，ｆ１∧ｆ２，ｆ１∨ｆ２，○ｆ，□ｆ，◇ｆ，ｆ１Ｕｆ２はＰＴＬ論理式．

記号¬，∧，∨は通常の論理式と同じ意味である．時制オペレータ，○，□，◇，Ｕの直観的意味は，

○ｆ　　…　ｆは次の状態で真，

□ｆ　　…　ｆは将来の全ての状態で真，

◇ｆ　　…　ｆは将来のある状態で真，

ｆ１Ｕｆ２…　ｆ2が真となる最初の状態までf1が真である．

このＰＴＬを用いて，並列システムの構造や制約を記述しようとするとき，次の３点で不便であった．

①状態の記述

並列システムの各要素ｅがそれぞれいくつかの有限な状態s_1,....,s_nをとるような構造を考える．ｅのある時点でとりうる状態はそのうち１つである．たとえば電灯のスイッチは，状態としてonとoff をとりうるが，各時点では排他的にどちらか一方の状態だけをとる．ｅの状態がs_iであるという命題をｅ（s_i）で表すと，ＰＴＬではｅが状態ｓｉであることを記述するために，

e（s_i）のほかに，s_i 以外の全ての状態 s_j に対して，¬e（s_j）であることを記述する必要がある．

②作用の記述

ＰＴＬでは，“ある要素ｅの状態が s_i である”と，“ｅの状態を s_i にする”をそれぞれ独立した命題として扱ことができる．しかし，このときｅを s_i にすれば，ｅは s_i になるという自明の関係を逐一記述する必要がある．

③状態保持の記述

ＰＴＬの各命題は時間毎に異なる真偽値をとりうる．要素ｅの状態が変化せずに保持される場合は，$e(s_i) \supset \bigcirc e(s_i)$ のようにＰＴＬで陽に記述しなければならない．大規模な並列システムになればなるほど，ある時間に変化する部分はごく一部である．しかし，ＰＴＬではその他の膨大な変化しない部分に対しても，変化しないことを記述しなければならない．

①②③ともに，記述する人間にとって大きな負担である．要素のとりうる状態数が大きくなればなるほど，記述量は爆発的に増える．これらは典型的なフレーム問題である．そこで，これらの問題点を解決を目的として，手順生成のための仕様記述言語ＰＴＳを設計した．

２．２　ＰＴＳの基本概念

（１）オブジェクトと状態

ＰＴＳのモデルは全てオブジェクトから構成される．各オブジェクトの内部状態は有限である．オブジェクトのとりうる状態の全体集合をドメインと呼び，

domain(<オブジェクト名>, <状態のドメイン>)

のように宣言する．ドメインをＤで表わす．

例：　domain（哲学者,[思考中，空腹，食事中]）.

原子式はブール式であり，次の形式である．

＜オブジェクト名＞（＜状態＞）

たとえば，原子式ｏｂｊ（s_1）は，モデルにおけるオブジェクトｏｂｊの状態がｓ１ならば，この式の評価値は真(true)であり，そうでなければ偽(false) である．ＰＴＳ式は原子式を∨，∧，¬によって組み合わせたものである．この場合の式の値は命題論理式と同様に定まる．

例：　哲学者1(食事中) ∧ ¬哲学者2(空腹)

通常の原子命題Ｐも次のように扱うことにより原子式で表すことができる．

ｄｏｍａｉｎ（Ｐ，［ｔｒｕｅ，ｆａｌｓｅ］）

Ｐ　ｉｆｆ　Ｐ（ｔｒｕｅ）

¬Ｐ　ｉｆｆ　Ｐ（ｆａｌｓｅ）

また，次のマクロ記法を導入する．

obj([s1,s2, …,sn])　ｉｆｆ

obj(s1) ∧obj(s2) ∧ … ∧obj(sn)

ここで，ｏｂｊ（Ｓ）の値は，オブジェクトｏｂｊの状態ｓが状態集合Ｓ＝[s1,s2, …,sn]に属しているならば，真であり，属していなければ偽である．さらに，

¬ｏｂｊ（Ｓ）＝ｏｂｊ（Ｄ－Ｓ）

ｏｂｊ（[]）＝ｆａｌｓｅ

ｏｂｊ（Ｄ）＝ｔｒｕｅ

（Ｄはドメイン，Ｓ⊂Ｄ，[] は空集合）

である．

（２）時間と時制オペレータ

各オブジェクトの状態は時間とともに変化する．時間は現在から無限の未来へ離散的に進む．時間にまたがる仕様を記述できるように，ＰＴＬと同じ直観的意味を持つ時制オペレータ○，□，◇，Ｕを導入する．

（３）作用オペレータと状態保持

ＰＴＳでは，“対象ｏｂｊの状態はｓである”（状態）と“対象ｏｂｊの状態をｓにする”（作用）という２つの命題を明確に区別する．たとえば，“スイッチはｏｎである”と“スイッチをｏｎにする”を別々の命題とする．この２つの命題には相互関係が定義できる．まず，作用オペレータ↑を導入し，次のように表記する．

obj(s)　：対象ｏｂｊの状態がｓである．

↑obj(s)　：対象ｏｂｊの状態をｓにする．

ここで，↑ｏｂｊ（ｓ）を“ｏｂｊ（ｓ）を真とする作用が存在する”と読む．↑ｏｂｊ（ｓ）はＰＴＳ式の原子式（つまり命題）である．このとき，作用の結果は次の時点で現れると考えるので，

□（↑ｏｂｊ（ｓ）　⊃　○ｏｂｊ（ｓ））

が成り立つ．

作用が存在しないときは，↑ｏｂｊ（ε）が真であると定義する．つまり，↑ｏｂｊ（ε）が真のとき，その時点でのオブジェクトｏｂｊの状態ｓは，次の時点でも保持される．すなわち，

□（obj(s)∧↑obj(ε) ⊃ ○obj(s)）

が成り立つ．ここで，ｏｂｊのドメインをＤとするとき，↑ｏｂｊのドメインD^+は，Ｄ∪｛ε｝であり，

↑ｏｂｊ（ε）　ｉｆｆ　¬↑obj_1（Ｄ）

である．

３．ＫＰＴＳ

ＰＴＳの核の部分（ＫＰＴＳ：Kernel of PTS ）に関して，厳密なシンタックスとクリプケ風セマンティクスを与える．また，ＰＴＬとの関係を示す．

3．1　KPTSの定義

［1］シンタックス

① 記号

オブジェクト：　obj1，obj2

状態：　s_1，s_2，……，s_n，ε

ドメイン：

$[s_1, s_2, \cdots\cdots, s_n] = D$

$[s_1, s_2, \cdots\cdots, s_n, \varepsilon] = D^+$

作用オペレータ：　↑

論理記号：　¬，∧，∨，⊃

時制オペレータ：　□，◇，○，U

② 式

原子項

D：ドメイン

s：状態

obj：オブジェクト

s∈Dのときobj（s，D）は原子項．通常，Dはオブジェクトに対し，一意に定まるのでobj（s）と略記する。

原子式

原子式は，状態を表わす状態原子式と，状態への作用を表す作用原子式による２ソート構造をとる．

obj（s，D）が原子項のとき

obj（s，D）は状態原子式

↑obj（s，D^+）は作用原子式

ここで，$D^+ \equiv D \cup \{\varepsilon\}$とする．

式

(1) 原子式は式

(2) f_1，f_2が式のとき，それらを論理記号または，時制オペレータで結合したものは式である．

注　$obj(s_1) \vee obj(s_2) \vee \cdots obj(s_n)$を省略して$obj([s_1, s_2, \cdots s_n])$と表記できる．

［2］セマンティクス

KPTS式のセマンティクスは構造M＝（W，N，π）で与えられる．

(1) Wは世界の可算集合

(2) N：W→Wは次の世界を与える関数

(3) $\pi : W \rightarrow \prod_{obj \in OBJ} (D_{obj}, \{0, 1\})$は，Wの各世界におけるオブジェクトの状態とオブジェクトへの作用の有無を与える関数．OBJはKPTS式に現れる全てのオブジェクトの集合であり，D_{obj}はオブジェクトobjのドメインである．便宜的に状態を与える関数をπ_1（w，obj）∈D_{obj}，作用の有無を与える関数をπ_2（w，obj）∈｛0（作用無），1（作用有）｝と表す．

N^i（w）をw　に続く世界の列

w，N（w），N（N（w）），……

のi＋1番目の世界とする．このとき

$<M, w> \models obj(s)$　iff　$\pi_1(w, obj) = s$

$<M, w> \models \uparrow obj(s)$　iff

s≠εのとき

$(\pi_1(w, obj) = s \supset$

$(\pi_2(w, obj) = 1$

$\wedge <M, N(w)> \models obj(s)))$

$\wedge (\pi_1(w, obj) \neq s \supset$

$<M, N(w)> \models obj(s))$

s＝εのとき

$\pi_2(w, obj) = 0$

$\wedge (\pi_1(w, obj) = s \supset$

$<M, N(w)> \models obj(s))$

$<M, w> \models \neg f$　iff　not $<M, w> \models f$

$<M, w> \models f_1 \wedge f_2$　iff

$<M, w> \models f_1$　かつ　$<M, w> \models f_2$

$<M, w> \models f_1 \vee f_2$　iff

$<M, w> \models f_1$　または　$<M, w> \models f_2$

$<M, w> \models \bigcirc f$　iff　$<M, N(w)> \models f$

$<M, w> \models \Box f$　iff

$(\forall i \geq 0)(<M, N^i(w)> \models f)$

$<M, w> \models \Diamond f$　iff

$(\exists i \geq 0)(<M, N^i(w)> \models f)$

$<M, w> \models f_1 \mathrm{U} f_2$　iff

$(\forall i \geq 0)(<M, N^i(w)> \models f_1)$

または

$(\exists i \geq 0)(<M, N^i(w)> \models f_2) \wedge$

$\forall j (0 \leq j < i \supset <M, N^j(w)> \models f_1)$

後半はPTLの構造と同じである．

ここで，任意の<M，w>を解釈とするとき，<M，w> ⊨ fを満す解釈<M，w>をfのモデルと呼ぶ．fにモデルが存在するとき，fは充足可能であるといい，存在しないとき，充足不可能であるという．

3．2　PTLとKPTSの関係

（定義）

KPTS式からPTLの論理式への変換tが等価変換であるとは，任意のKPTS式fが充足不可能ならば，PTLの論理式t(f)も充足不可能であり，かつfが充足可能ならば，t(f)も充足可能であることである．PTLからKPTSへの変換の等価性についても同様に定義する．

（補題１）

ＫＰＴＳの原子式をＰＴＬの原子論理式とみなし，次の前提条件を追加する変換ｔは，ＫＰＴＳからＰＴＬへの等価な変換である．

$$P \equiv \bigwedge_{obj_i \in OBJ} (Premise1(obj) \wedge Premise2(obj))$$

ここで，ＯＢＪはＫＰＴＳ式に現れる全てのオブジェクトの集合である．また，Premise1とPremise2は次のように定義する．

①単一状態条件

オブジェクトｏｂｊは各時点で，ドメインＤのうちのただ１つの状態をとる．

$$Premise1(obj) \equiv$$
$$\Box \{ \ (\bigvee_{s \in D} obj(s))$$
$$\wedge \neg (\bigvee_{s,s' \in D, s \neq s'} (obj(s) \wedge obj(s')))$$
$$\wedge (\bigvee_{s \in D^+} \uparrow obj(s))$$
$$\wedge \neg (\bigvee_{s,s' \in D^+, s \neq s'} (\uparrow obj(s) \wedge \uparrow obj(s'))) \}$$

②作用条件

オブジェクトｏｂｊのドメインをＤとするとき，ＫＰＴＳの作用と状態保持は，次のＰＴＳの前提として表すことができる．

$$Premise2(obj) \equiv$$
$$\wedge \{\Box ((obj(s) \wedge \uparrow obj(\varepsilon)) \supset \bigcirc_{s,s' \in D} obj(s))$$
$$\wedge \Box ((\uparrow obj(s') \supset \bigcirc obj(s'))) \}$$

（補題２）

ＰＴＬ式ｆの全ての原子命題ｐをＫＰＴＳ式にの作用原子式↑ｐ(true,[true,false]) に置き変える変換ｔは，ＰＴＬからＫＰＴＳへの等価な変換である．

（定理）

任意のＫＰＴＳ式はＰＴＬ式に等価変換可能である．また，その逆も可能である．

４．ＰＴＳ

ＫＰＴＳは，記述言語としての本質的特徴をよく表しているが，そのままではユーザにとって使い易いとはいえない．ＰＴＳは，ＫＰＴＳを記述容易さ，および効率の観点から洗練化された仕様記述言語である．

４．１　ＰＴＳの定義

ＫＰＴＳにメソッド記述と局所手順記述を導入し，逆に式の中に直接作用を記述を禁止した仕様記述言語をＰＴＳと呼ぶ．ＰＴＳのシンタックスを次に示す．

＜ＰＴＳ仕様＞：：＝
　　［　　＜ドメイン宣言＞，
　　＜メソッド記述＞，
　　＜局所手順集合＞，
　　＜制約条件＞　　］．
＜ドメイン宣言＞：：＝
　［domain(<オブジェクト名>，<状態集合>)，…］
＜メソッド記述＞：：＝
　　［＜メソッド名＞，＜メソッド＞），…］
＜局所手順集合＞：：＝
　　［＜局所手順＞，＜局所手順＞，……］
＜制約条件＞：：＝
　　［＜ＫＰＴＳ式＞，…］

ここで，ドメイン宣言はＫＰＴＳと同じである．制約条件のリストの要素は，作用を含まないＫＰＴＳ式である．＜メソッド記述＞と＜局所手順記述＞については，次の節以降で定義する．

４．２　メソッド

各時点で同時に存在する作用をメソッドとしてまとめる．作用は基本的要素であるが，手順仕様記述においては，複雑な作用の組み合せは必ずしも必要でない．そこで，作用を直接開放せずに，メソッドという一種のマクロ表現を提供することにより，作用の使用を限定する．メソッドの導入により，記述は直観的に把握しやすくなり，生成手続きは高速化できる．メソッドの定義を示す．

＜メソッド＞：：＝
　　ｍｅｔｈｏｄ（＜メソッド名＞，
　　＜ガードリスト＞，
　　＜作用リスト＞）
＜ガードリスト＞：：＝　［＜状態原子式＞，…］
＜作用リスト＞：：＝　［＜作用原子式＞，…］

ガードリストと作用リストにおいて，［＜式＞，＜式＞，…］は＜式＞∧＜式＞∧……を表す．作用リスト中の作用原子式のオペレータ↑は省略される．

例えば，ｍｅｔｈｏｄ（ｍｋ，ｇｌ，ａｌ）において，ｍｋはメソッド名，ｇｌはガードリスト，ａｌは作用リストであり，ｇｌに属する状態原子式が真のときのみメソッドｍｋ が起動でき，その結果，作用リストａｌの各

作用が実行される．

厳密には，メソッドはＫＰＴＳのマクロ表現として定義されるが，ここでは省略する．このとき便宜上，各メソッドは，ｍｅｔｈｏｄというオブジェクトのとる状態として定義される．

メソッドの特徴を示す．

①いくつかの作用をまとめたものがメソッドである．

②各時点で必ず１つのメソッドだけが実行される．

③作用リストに記述されていないオブジェクトについては，作用が行われなかったとする．

④メソッドhaltは，それ以後作用がおこらず変化が停止することを意味する．

メソッドの記述の導入により，ある種のフレーム問題は解決されている．つまり，ユーザは変化させたい対象だけ記述すればよく，記述されない対象はデフォルトとしてその時の状態を保持する．

４．３　局所手順記述

局所的に見て決定できるメソッドの具体的順序関係を，局所手順記述として与える．局所手順は複数あってよい．これは，並列システム中の要素の局所的動作の構造を表現している．各局所手順をプロセスであるとみなせば，局所手順の集合は並列プログラムと解釈することもできる［Pnueli81］．局所手順集合Ｐは，局所手順p_i（$1 \leq i \leq n$）のリストで表す．

$$P = [p_1, p_2, p_3, \cdots\cdots, p_n]$$

各局所手順p_iは，メソッド名をシンボルとする有限オートマトンで表す．

＜局所手順＞：：＝（Ｍ，Ｔ，s_0）

Ｓ　：状態集合

Ｍ　：局所メソッドの集合

Ｔ　：状態遷移規則　（trans:Ｓ×Ｍ×Ｃ→Ｓ）

Ｃ　：遷移条件

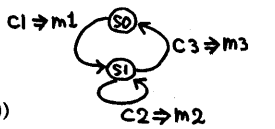
s_0：初期状態

ｔｒａｎｓ（ｓ１，ｓ２，ｃｌ，ｍ）＝ｓ２の意味は，状態ｓ１において，条件リストｃｌの各条件が成り立ち，メソッドｍのガードｇが真ならば，ｍを実行して状態ｓ２に遷移するということである．ある状態ｓ１において，全ての遷移が不可能ならば，その局所手順は状態ｓ１のままでサスペンドする．

局所手順は有限状態遷移グラフで表すと理解しやすい．グラフの各辺には条件ｃｌとメソッドm_iがラベルとしてつく．また，各ノードには状態名s_iがつく．そのうちの１つは初期状態である．

局所手順の例を示す．

```
([ trans(s0,s1,[c1],m1),
   trans(s1,s1,[c2],m2),
   trans(s2,s1,[c3],m3)],s0)
```



厳密には，局所手順はメソッドとＫＰＴＳ式のマクロ表現として定義できる．局所手順ｐの状態は，ＫＰＴＳ式のｐ([< 状態>])として表され，メソッドが実行される毎にオブジェクトの状態が変化する．初期状態もＰＴＳ式の制約条件として与える．すなわち，局所手順ｐの各遷移trans（s_i，s_j，cl_k，m_k）に対して，対応するメソッド

method（m_k，gl_k，al_k）を，

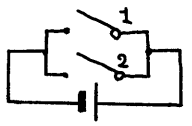
method（m_k，$p([s_i])\oplus cl_k \oplus gl_k$，$p([s_j])\oplus al_k$）

に変換する．また，初期状態は，ＰＴＳ式$p([s_0])$として，制約条件部に追加する．

ここで，⊕はリストの結合を表す．つまり，[a1,a2,a3,…,an] ⊕ [b1,b2,b3,…,bm]＝[a1,a2,a3,…,an,b1,b2,b3,…,bm]である．

４．４　記述例：スイッチの開閉手順

```
[   domain(スイッチ1,[on,off]),
    domain(スイッチ2,[on,off])          ],
[   method(入れる1,[スイッチ1([off])],[スイッチ1([on])]),
    method(切る1,[スイッチ1([on])],[スイッチ1([off])]),
    method(入れる2,[スイッチ2([off])],[スイッチ2([on])]),
    method(切る2,[スイッチ2([on])],[スイッチ2([off])])      ],
[],
[   スイッチ1([on]) & スイッチ2([off]),
    [](スイッチ1([on]) # スイッチ2([on]))                   ]
].
```



ここで　& … ∧
… ∨
[] … □
<> … ◇
ー … ¬
である

図１　スイッチの開閉手順

５．ＰＴＳによる手順の自動生成

メソッドの列を手順と呼ぶ．ＰＴＳで記述された仕様を満たす全ての手順を自動生成する手法を示す．

５．１　モデルの生成手続き

ＰＴＳ式を充足する全てのモデルを生成するアルゴリズムを示す．基本的な流れはＰＴＬのモデル生成手続きであるタブロー法［MW,Pla］と同じである．

ここでは，状態と作用に関して，タブロー法と異なる部分を中心に示す．

モデルの生成手続きは，分解手続き，削除手続き，および調整手続きから構成される．

（１）分解手続き

ＰＴＬに対するタブロー法の分解手続きは，分解ルー

ル（□f ⟶ f∧○□f，◇f ⟶ f∨○◇f）を基本に，任意の論理式F(t)を現在Pと次の時点以後の論理式Nに分解し，積和標準形にする（$F(t) \rightarrow \bigvee_i (P_i \wedge \bigcirc N_i)$）．次に，$F(t+1) = N_i$ として，分解操作を繰りかえし適用すれば，有限回で将来起こりうる全ての論理式のパターンを生成できる．

ＰＴＳではある時点における作用と状態の保持が，次の時点以後に影響を与えるので，分解は多少複雑になる．

任意のＰＴＳ式F(t)をまず現在(t)の式Pと次の時点(t+1)以後の式Nに分解し，積和標準形にする（$F(t) \rightarrow \bigvee_i (P_i \wedge \bigcirc N_i)$）．次に各$P_i$において適用可能なメソッドを抽出し，$M = \{m_1, \cdots, m_k\}$とする．各メソッド$m_j \in M$（ここで，method$(m_j, gl_j, al_j)$と定義されているとする）ごとに，$P_i \wedge gl_j$に対してメソッドに記述されている作用リスト$al_j$を実行し，作用が記述されていないオブジェクトに対しては状態を保持させた結果をI_{ij}とする．I_{ij}はメソッドの数だけ生成される．$F(t+1) = N_i \wedge I_{ij}$を次の時点の論理式として，再帰的に分解する．（図２）

F(t)をノードし，m_jをエッジとすると，F(t)のパターンは有限であることが保証されるので，分解の結果は，有限グラフで表すことができる．

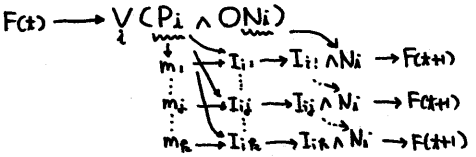


図２　ＰＴＳの分解手続き

（２）削除手続き

削除手続きにより，生成手続きで得られたグラフにおいて，◇fの形の論理式が，グラフ上で充足可能かどうかをチェックし，可能でなければ，その論理式のついたエッジを削除する．この手続きは，ＰＴＬのタブロー法と同じであり，詳細は［Plai86］にあるので，ここでは省略する．グラフの強連結成分を利用すると速い．

（３）調整手続き

削除手続き後のグラフが空でなければ，その論理式は充足可能である．決定手続としてはここで十分だが，モデルを生成する手続きとしては，状態の集合表現に関する後処理が必要である．つまり，エッジにobj([s1,s2,s3])とあったとき，必ずしもobj(s1),obj(s2),obj(s3)のそれぞれが真であるモデルが存在するとはかぎらない．そこで，真となりえない状態を検出し状態集合から削除しなければならない．これを調整手続きと呼ぶ．

生成された有限グラフをモデルグラフと呼ぶ．モデルグラフ上を公平に移動して，メソッドを実行する系列は，ＰＴＳ式を充足している．

生成手続きが高速化できる要因は，次の２点に要約できる．

①集合演算による高速化

状態原子式の否定形は，状態集合の補集合として表すことができ，ＡＮＤ，ＯＲは集合演算として計算できる．

②メソッドによる高速化

メソッドの単一起動条件を最大限利用した生成手続きになっている．また，メソッドの導入により，作用原子式を直接処理する必要がない．

５．２　手順の生成手続き

モデルグラフから手順を生成する．まず手順の定義を行う．

（１）手順の定義

手順はメソッドの列である．ＰＴＳにおけるメソッドは，具体的な手順の表現には不十分であるので，メソッドの中にPrologプログラムの記述を許すことにする．拡張されたメソッドは次のような形式をとる．

```
method (m, gl, al)
  ←pc1,pc1,…,pck  |  pa1,pa2,…,pal.
    m   ：メソッド名
    gl：ガードリスト
    al：作用リスト
    pc1, …, pck ：Prologの述語（第２次制約）
    pa1, …, pal ：Prologの述語（第２次作用）
```

すなわち，メソッドmは，glが真であり，かつ$pc_1, \cdots, pc_k$が真のときのみalの作用を実行するとともに，$pa_1, \cdots, pa_l$のProlog述語を実行する．

ＰＴＳにより記述された制約(gl)を第１次制約と呼び，Prologで記述された実行条件を第２次制約と呼ぶ．また，作用リストalを第１次作用とよび，実行されるPrologのゴール列を第２次作用と呼ぶ．拡張されたメソッド（すなわち手順）の第２次制約，第２次作用を無視したものがＰＴＳのメソッドである．第１次制約では，手順の仕様の骨格が記述される．第１次制約では，時制命題論理の範囲でしか記述できないので，それ以外の詳細な制約は，第２次制約として記述する．第２次制約はメソッド実行時に動的に評価されるので，ロボットの外部センサーによる制約の記述も可能となる．

第２次作用として，任意のPrologプログラムを実行できる．しかし，ＰＴＳで記述されたオブジェクトの状態を変化させることはできない．

つまり，システムの制約の骨格を第１次制約としてＰ

ＴＳで記述し，モデル生成手続きにより制約を満す手順の全てのモデルを生成する．細かい制約は第２次制約として記述し，実行時の手順の選択に適用する．

（２）　第１次制約と第２次制約の関係

手順が第１次制約／作用のみの場合は，実行時の不確定要素がないので，仕様を満す手順を１つでも生成すれば十分である．しかし，第２次制約が存在する場合，実行前に第１次制約／作用を満す全ての手順を生成しておく必要がある．実行時にその手順の集合の中から第２次制約を満たす手順が選択される．

（３）　第１次制約による可能手順グラフの生成

手順仕様のうち第２次制約，第２次作用を無視したものはＰＴＳ式である．このＰＴＳ式からモデルグラフを求め，これを可能手順グラフと呼ぶ．手順仕様の第１次制約／作用を満たす全ての手順は，この可能手順グラフ上に表わされる．

（４）　第２次制約による実行時手順選択

可能手順グラフ上を公平に移動しながら辺にラベルとしてついているメソッド実行すれば，少なくとも，第１次制約は満たされる．実行時には，可能手順グラフ上で実行可能なメソッドのうち，第２次制約を満たしたものを選択し実行する．その結果次のことが起こらなければ，実行された手順は仕様を満たしている．

①デッドロックをおこす．

②選択が公平でない（無限手順の場合のみ）

しかし，①，②が起きたからといって仕様に矛盾があったとはいえない．①は発生時点で検出できるが，②の検出は難しい．しかし，手順の最後にｈａｌｔメソッドがある場合，手順は有限であり，②は考えなくてよい．

６．記述量と計算量の評価

簡単な例で，ＰＴＬとＰＴＳの記述量と計算量を比較する．

例：ｋ個の状態を持つオブジェクトobj1とobj2がある．各オブジェクトの状態を順々に繰り返し遷移させるような手順を生成せよ．

この例でｋ＝２の場合のＰＴＳ記述は図３になる．ｋ＝１，２，…，６の場合のＰＴＬとＰＴＳの記述量と計算量を図４に示す（注１）．ＰＴＬとＰＴＳの処理系は別個に実装されているので，単純には比較できないが，だいたいの傾向はつかめる．問題の大きさ（ｋ）が大きくなるにつれて記述量，計算量ともに増加するが，ＰＴＳの増加率はＰＴＬよりもはるかに穏やかである．一般には，ＰＴＳがＰＴＬより効率の悪い例も存在するが（注２），多くの実際的例では，ＰＴＳのほうが実用的である．

注１　記述量は文献［Clar86］の論理式の長さの拡張で，式の構造を２分木で表現したときのノード数を表す．計算量はｃｐｕタイムとする．

注２　ＰＴＬ式が，ドメインが［true,false］のＰＴＳ式にそのまま対応している場合．

```
[
  [ domain(obj1,[s1,s2]), domain(obj2,[s1,s2]) ],
  [ method(m1,[obj1([s1])],[obj1([s2])]),
    method(m2,[obj1([s2])],[obj1([s1])]),
    method(n1,[obj2([s1])],[obj2([s2])]),
    method(n2,[obj2([s2])],[obj2([s1])]) ],
  [ obj1([s1]) & [](<>obj1([s1])) & [](<>obj1([s2])) &
    obj2([s1]) & [](<>obj2([s1])) & [](<>obj2([s2])) ]
].
```

図３　ＰＴＳ記述（k=2 の場合）

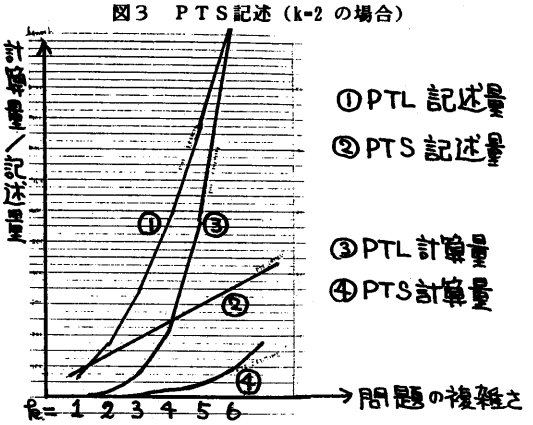


図４　ＰＴＬとＰＴＳの比較

７．例：電力系統操作手順生成

（１）問題定義

電力系統における平常時の母線切替操作手順生成にＰＴＳを適用する．この問題の出典は文献［松本84］である．図５に示す変電所モデルにおいて，母線１から母線２への切替を行う手順を生成したい．このときの制約条件は，次の２点である．

①常に電気は流れている（無停電条件）．

②断路器による負荷電流の開閉がない（断路条件）．具体的には，断路器は対応する遮断器がoff であるか，遮断器を通る分路があればoff にできる．

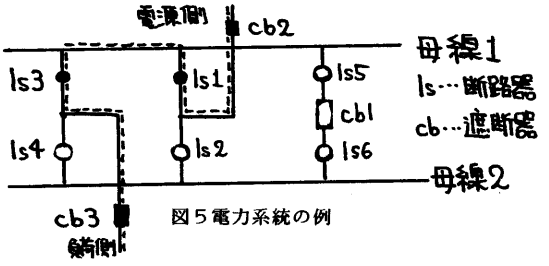


図５電力系統の例

（2）ＰＴＳによる仕様記述

①ＰＴＳ（局所手順なし）

ＰＴＳによる記述を図６に示す．断路器，遮断器共にドメインは｛on，off｝である（*1）．各開閉器のon，off 操作をメソッドとして記述する．断路条件は，メソッドの第１次制約として記述される（*2）．無停電条件としては，常にいずれかの経路が通電可能であることを記述する（*3）．初期状態として，母線１により電気が流れていること（*4），ゴールとして，母線２により電気が流れていること（*5）を記述する．第２次制約としては，ある開閉器を操作したら過途電流が安定するまで，次の操作の実行を待つといった記述があるが，ここでは省略している．

②ＰＴＳ（局所手順あり）

①のＰＴＳの仕様記述に対し，次の３つの局所手順を追加することができる．

a) ｃｂ１，ｌｓ５，ｌｓ６に対する局所手順（図6a）

b) ｌｓ１，ｌｓ２に対する局所手順（図6b）

c) ｌｓ３，ｌｓ４に対する局所手順（図6c）

（3）手順生成結果

ＰＴＳ（②）から，第１次制約／作用により可能手順グラフが生成できる（図７）．このグラフの任意のパス（例えば，ls5on-ls6on-cb1on-ls2on-ls1off-ls4on-ls3off-cb1off-ls5off-ls6off-halt ）は仕様を満たす手順である．ここで，ＰＴＳ（①，②），ＰＴＬ（①と対応するＰＴＬ論理式），ＰＴＬ（［松本84］にあるＰＴＬ論理式）による記述量と生成時間を表１にまとめた．

［松本84］では，手順の検証にＰＴＬを用いているので，手順の記述が陽に書かれているわけではないが，内容的にはＰＴＳ①と等価である．

	ＰＴＳ ①	ＰＴＳ ②	ＰＴＬ ①対応	ＰＴＬ ［松本84］
記述量	905	862	1873	691
計算量 (cpu秒)	760	145	＊＊	283

表１　電力系統の記述量，計算量

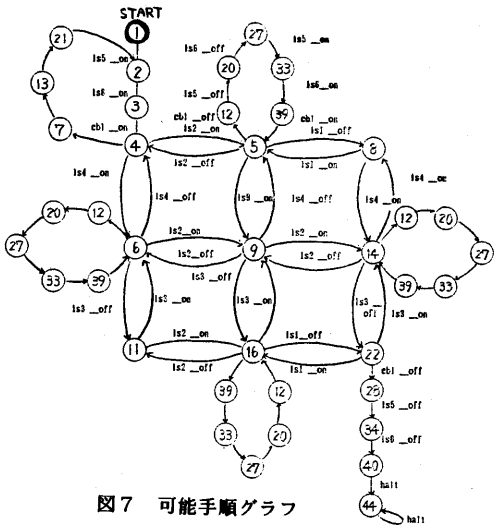


図７　可能手順グラフ

```
[
 [   % *** ドメイン宣言 ***
   domain(ls1, [on, off]),  domain(ls2, [on, off]),
   domain(ls3, [on, off]),  domain(ls4, [on, off]),
   domain(ls5, [on, off]),  domain(ls6, [on, off]),
   domain(cb1, [on, off]),  domain(cb2, [on, off]),
   domain(cb3, [on, off]) ],                                                    } *1
 [   % *** メソッド宣言 ***
   method(ls1_on, [ls1([off]), ls2([on]), cb1([on]), ls5([on]), ls6([on])],  [ls1([on])] ),
   method(ls1_off, [ls1([on]), ls2([on]), cb1([on]), ls5([on]), ls6([on])],  [ls1([off])] ),
   method(ls2_on, [ls2([off]), ls1([on]), cb1([on]), ls5([on]), ls6([on])],  [ls2([on])] ),
   method(ls2_off, [ls2([on]), ls1([on]), cb1([on]), ls5([on]), ls6([on])],  [ls2([off])] ),
   method(ls3_on, [ls3([off]), ls4([on]), cb1([on]), ls5([on]), ls6([on])],  [ls3([on])] ),
   method(ls3_off, [ls3([on]), ls4([on]), cb1([on]), ls5([on]), ls6([on])],  [ls3([off])] ),
   method(ls4_on, [ls4([off]), ls3([on]), cb1([on]), ls5([on]), ls6([on])],  [ls4([on])] ),
   method(ls4_off, [ls4([on]), ls3([on]), cb1([on]), ls5([on]), ls6([on])],  [ls4([off])] ),
   method(ls5_on,  [ls5([off]), cb1([off])], [ls5([on])]),
   method(ls5_off, [ls5([on]), cb1([off])], [ls5([off])]),
   method(ls6_on,  [ls6([off]), cb1([off])], [ls6([on])]),
   method(ls6_off, [ls6([on]), cb1([off])], [ls6([off])]),
   method(cb1_on, [cb1([off])], [cb1([on])] ),
   method(cb1_off, [cb1([on])], [cb1([off])] ),
   method(cb2_on,        [cb2([off])], [cb2([on])]),
   method(cb2_off,       [cb2([on])], [cb2([off])]),
   method(cb3_on,        [cb3([off])], [cb3([on])]),
   method(cb3_off,       [cb3([on])], [cb3([off])])                           } *2
 ],
 [  % *** 初期条件 ***
   ls1([on]) & ls2([off]) & cb1([off]) & ls3([on]) & ls4([off]) &
     cb2([on]) & cb3([on]) & ls5([off]) & ls6([off]),                           } *4
   % *** ゴール ***
   <>( ls2([on]) & ls1([off]) & cb1([off]) & ls4([on]) & ls3([off])
     & cb2([on]) & cb3([on]) & ls5([off]) & ls6([off])),
   []( ( ls2([on]) & ls1([off]) & cb1([off]) & ls4([on]) & ls3([off])
     & cb2([on]) & cb3([off]) & ls5([off]) & ls6([off])) => method([halt])),    } *5
   %*** 無停電条件 ***
   []( (cb2([on]) & ls1([on]) &  ls5([on]) & cb1([on]) & ls6([on]) & ls4([on]) & cb3([on]) ) #
     (cb2([on]) & ls2([on]) &  ls5([on]) & cb1([on]) & ls6([on]) & ls3([on]) & cb3([on]) ) #
     (cb2([on]) & ls1([on]) & ls3([on]) & cb3([on])) # (cb2([on]) & ls2([on]) & ls4([on]) & cb3([on])
     ) )                                                                        } *3
]
].
```

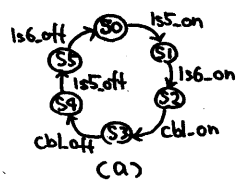


図６　PTSによる仕様記述

（4）考察

記述面：　ＰＴＬだけによる記述より，作用が陽に表現できるので直観的にわかりやすく，記述量も少ない．また，ops5等を用いたエキスパートシステム的手順生成手法に比べて、

①生成された手順の正当性が論理的に保証されている．

②状態の制約条件と同じレベルでメソッド間の相互制約を記述できる．よって，その混合記述ができる（ops5では，メソッド（ルール）間の相互制約は，LEX/MEA戦略として別のレベルで扱われる）．

などの利点がある．②について電力系統の場合は，ある開閉器をｏｆｆ（ｏｎ）にした直後に同じ開閉器をｏｎ（ｏｆｆ）にして元の状態に戻すような無駄な操作は行わないというメソッド間制約が記述可能である．

例：□¬(method(isl(on))⊃method(isl(off)))

効率面：　ＰＴＬ式（①対応）の計算は，１日で終了しなかった（SUN3,CPROLOG）．ところが，［松本84］のＰＴＬ式は，ＰＴＳ（①）より効率がよい．これは，各開閉器のon,offをそれぞれ真，偽で表現するなど，かなり最適化された論理式だからである．しかし，状態数が３以上になると真，偽で表なくなるので，ＰＴＳの効果がより期待できる．局所手順の効果も顕著（５倍）である．

８．まとめ

手順の自動生成のための仕様記述言語ＰＴＳを提案した．ＰＴＳはＰＴＬに比べ，次の特徴がある．

① 言語の記述能力はＰＴＬと同等であるが，手順生成のための仕様記述言語としての記述容易性ははるかに上である．

② ＰＴＳの言語の特徴を生かした効率的な生成手法が存在する．

③ あらかじめわかっている局所的な手順を明示的に仕様の中に記述することができ，生成時の組み合せ的爆発をおさえるのに効果的である．

以上の特徴により，従来では対処できなかった複雑さを持つ現実の問題に対しても，ＰＴＳを用いた時制論理的アプローチによる手順生成がある程度適用可能になった．

現在，ＰＴＳをＦＡの分野に適用し，評価，改良を行なっている．ここで，制御不可能な外界からの作用を考慮した手順生成を目的としてＰＴＳを拡張したＰＴＳ＋＋［川田88］を開発した．また，並列プログラミング言語ＭＥＮＤＥＬ88［内平88］の同期部の仕様記述言語としてＰＴＳを採用し，プログラムの自動生成手法を検討している．

謝辞

本研究の一部は，ＩＣＯＴの再委託研究の一環として行なわれた．ＩＣＯＴの関係各位に深謝する．また，有益なご意見をいただいた東芝の中村英夫氏，本位田真一氏，西村一彦氏，伊藤美香子氏に感謝する．

参考文献

[Allen83]J.Allen,Maintaining Knowledge about Temporal Intervals,CACM,Vol.26,No.11,1983.

[Clar86]E.M.Clarke et.al.,Automatic Verification of Finite State Concurrent System Using Temporal Logic Specification,ACM TOPLAS,vol.8,No.2,1986.

[Pusa83]A.Fusaoka et.al.,A Description and Reasoning of Plant Controller in Temporal Logic,8th IJCAI 1983.

[GN87] Genesereth,M.R.,Nilsson,N.J.,Logical Fundations of Artificial Intelligence,Morgen Kaufmann Publishers,pp263-305,1987.

[Kowa86]R.Kowalski,M.Setgot,A Logic-based Calculus of Event,New Generation Computing 4, 1986.

[Pnueli81] Pnueli,A.,The Temporal Semantics of Concurrent Programs,Theore. Compu. Sci. 13,1981.

[Stu85] Stuart,C.,An Implementation of Multi-Agent Plan Synchronizer,IJCAI85,1985.

[Uchi87] Uchihira,N.,Kasuya,T.,Mastumoto,K., and Honiden,S.,Concurrent program synthesis with reusable componts using temporal logic,Proc. of COMPSAC 87,1987.

[MW84] Manna,Z. and Wolper,P.,Synthesis of communicating processes from temporal logic specification, ACM TOPLAS,Vol.6,No.1,1984.

[内平88] 内平　他，ペトリネットと時制論理による並列プログラミング言語，情報処理研究会17-2,1988.

[川田87] 川田，内平，　他，時制論理ベースの形式的仕様記述言語ＰＴＳ，日本ソフトウェア科学会第４回大会 p251-254,1987.

[川田88] 川田，内平，時制論理に基づく仕様記述言語ＰＴＳ＋＋による並列システムの制御，日本ソフトウェア科学会第５回大会,1988.

[佐伯] 佐伯，非単調時制論理とその形式的仕様記述への応用，情報処理学会論文誌，vol.28,no.6,1987.

[西村87] 西村，溝口，時間概念の表現とその応用，日本ソフトウェア科学会第４回大会 p211-214,1987.

[丸田87] 丸田　他，通信プロトコルの時間的制約に関する知識表現の検討，情報処理第35回全国大会,1987.

[松本84] 松本，坂口，系統操作の正当性を自動検証する一手法，電気学会論文誌59-B74,1984.