7M-1

# 時制論理に基づく仕様記述とそのデバッグ環境

西村一彦　内平直志
(株)東芝　システム・ソフトウェア技術研究所

## 1.はじめに

並列プログラムの実行過程の解明や知能ロボットの行動計画生成問題は時間的関係を含んだ問題として定式化され，その方法として時制命題論理(PTL:Propositional Temporal Logic)による解決が試みられている．[1][4]

このようなPTLをベースとした具体的システムでは，論理式を形式的な仕様とみなし，仕様を満たすような手順系列を自動的に生成することができる．また，PTLで書かれた仕様が正しければ，その結果得られる手順系列も正しいことが保証されている．しかし，PTLによって記述された仕様（論理式）自体に誤り，不足がある場合は当然のことながらその結果も誤りとなる．ところが，結果の検証，デバッグはいまだに人手で行なわれている．バグとしては

(1)記述ミス（スペルミス）
(2)きつい仕様（無駄な制約）
(3)仕様が記述不足（制約が不足）

を考える．(1)は原子命題や論理記号の記述ミスである．(2)では，無駄な制約のために，結果として得られるモデル集合(M)がユーザーが実行したいもの(U)に比べ小さなもの，すなわち，M⊆Uとなる．従って，MをUによって検証すれば必ず矛盾が発生し，その矛盾点を追跡することでバグを発見することができる．ところが，(3)は必ずしも検証過程で矛盾を引き起こすとは限らない．なぜなら，M⊇Uだからである．この場合，むしろ，不足している仕様が何であるかをユーザーから与えられるモデルから生成する機能が必要となる．そこで，(3)以外の二つのバグを同定する方法について述べる．

まず，ここで対象とするPTLとその決定手続きであるタブロー法について解説し，PTL式のバグが何であるのかを定義する．次にユーザーモデルに基づいたバグの発見アルゴリズムついて詳述する．

## 2.ＰＴＬ

PTL[1]は命題論理に○(next)，◇(eventually)，□(always)，∪(until)といった時制演算子を追加し，時間関係を陽に扱うための論理体系である．特にここでは線形時制命題論理(Linear PTL)を対象としている．

充足可能性の決定手続きはタブロー法を用いて行なわれ，与えられた論理式の集合を，(1)分解手続き，(2)グラフ作成部，(3)削除手続きの３つのステップによって証明を行なう．その概要を以下に示す．

(1)与えられた論理式の集合 Sに含まれる任意の式Fを，分解規則に従って現在とそれ以後に分解する．これは現在はただ一つの原子命題が真であり，それ以外は偽であるとするsingle event conditionを前提としている．分解の終了した論理式 Fをマークする．この作業を全ての論理式がマークされるか，もしくは○オペレータで囲まれた論理式となるまで行う．

(2)与えられた論理式の集合を初期ノードとし，(1)の操作を再帰的に適用する．そして得られた論理式の集合（マークされた式か○ー式）と現在唯一真である原子命題を辺とし，○で囲まれた式からなる論理式の集合を新たなノードとして，グラフを構築する．

(3)(2)で得られたグラフからeventualityが満たされていない辺を削除していく．その結果は与えられた仕様を満たす完全なモデル集合となる．以後，このグラフをモデルグラフ(M)と呼ぶ．

モデルグラフは状態遷移図で表現される．（図１）

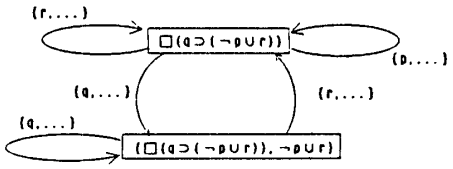


図１　充足可能なPTL式（[1]より）

## 3.バグと症状

PTL式で与えられた仕様が先に述べたバグ（記述ミスと無駄な制約）を含んでいる場合，その症状としては次のものが考えられる．

(1)モデルがない(M=φ)　(S-1)
(2)異なるモデル集合(M')が得られる　(S-2)

(1)は与えられた論理式が充足不可能であることを意味しており，仕様中に矛盾がある場合にはこのような症状を発生する．その原因は，明らかに論理式の間に矛盾があるか，もしくはeventuality が満たされない場合である．

(2)は仕様に矛盾はないが，ユーザーの意図したものとは異なったモデルができてしまう．これを発見するには得られたモデルを検証しなければならない．そこで動的な検証（トレース）を行いながら，バグの発見を行なう方法について述べる．

## 4.デバッグ方法

### 4.1証明過程の保存

仕様のバグの発見を容易にするために PTL式の証明過程を可視化し，保存する必要がある．

可視化する際，どの程度の情報量を与えれば良いのかが問題となる．本稿で対象としたPTL式の証明は，single event condition を前提としているので，論理式の集合（仕様）中の任意の論理式について分解は逐次的に行われ，しかも，分解の結果は他の論理式に影響しないことから，分解が行われた部分（変化した部分）だけを示せば良い．そして，以下の形で証明過程を保存する．

---

The Debugging Environment for Specification Language based on PTL
Kazuhiko NISHIMURA, Naoshi UCHIHIRA
TOSHIBA Corp. Systems and Software Engineering Laboratory

ptree(分解ノード,
　　生成ノード,
　　現在真である原子命題,
　　分解論理式).

述語ptree はノードが論理式中の全ての原子命題に対してどのように展開されたか，また展開の結果を示す証明木を構築する．（図２）

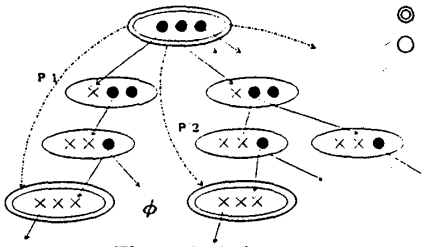


図２　証明木

4.2　モデルグラフの検証

(S-2)の場合，生成されたモデルグラフ(M) がユーザーの意図したものかどうかを検証する必要がある．M は状態遷移図の形で得られることから状態遷移図上でユーザーの意図したもの（実行可能な手順系列）が成立することを調べる．この手順系列は，実行可能なものであり，本稿では実行不可能な手順系列は考えていない．そして，この系列はアトム列で与えられ，これをユーザーモデル(U)として定義する．

U は有限系列に限っている．ただし，ループを含んでもよい．U を満たすものがあれば仕様にバグがないものとする．そうでなければ次節以下のデバッグを行う．

4.3 デバッギングアルゴリズム

基本的に証明木を追跡することによってバグの発生箇所をつきとめていく．その方法は症状によって異なる．

(1)静的解析

U にM の中に現われない原子命題があるとき，その原子命題を提示する．

(2)(S-1)の場合

入力：空なるモデル(M=φ)を構築する論理式の集合
出力：矛盾する論理式（集合）
手順：初期ノードから，分解手続きを段階的に実行する．ここで，新しいノードが生成されなければ（グラフが構築されない）証明木の末端はすべて空となる．（図３）

証明木から空ノードを導く過程を各々について示し，どの展開が間違ったかをユーザーに問い合わせることによってバグを同定する．そうでなければ，eventuality のチェックを行ない，各辺上で満たされていないeventuality を提示し，この作業を終了．

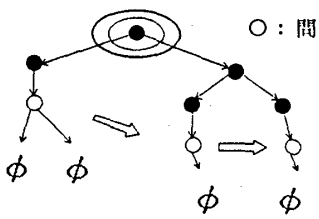


図３　空モデルに対するデバッグ

(3)(S-2)の場合

入力：異なるモデル集合(M')を生成する論理式の集合
出力：U において，最初に満たされない手順とその時の状態を示し，その状態を生成した論理式を提示する．
手順：初期ノードS0から始まり，U が満たされているかどうかをチェックする．満たされない手順P∈Uが発見されたら，チェックを止める．そのときの状態をS とする．そして，以下の作業をS0に至るまで行う．

Sにおいて Pが唯一真であるとすると，それに矛盾する論理式 Fが存在する．F が唯一の場合，それがどの論理式から導かれたのかは証明木を直接追跡することで提示できる．

F が複数に及ぶ場合は任意の一つを選んで初期ノードに向かって段階的に追跡していく．ユーザーは各段階でシステム側からの質問に答える．質問は失敗する手順について，その分解が正しいかどうかについてである．もし，その分解が正しくなければ，

F を導いた仕様を追跡し，提示する．また，正しければ，他のF について同様な質問をする．F がすべて正しければ，P の一つ前の手順についてその分解が正しいかどうかを調べる．最終的に，ユーザーモデルすべてについてこの作業を繰り返す．

5.まとめ

本稿では時制論理に基づいたプログラミング環境を充実させるためにデバッグの方法について述べた．本方式では特に，無駄な制約のために起こる失敗に対して，有限な手順系列を与え，その仕様から得られるモデルグラフ（状態遷移図）を動的に検証しながらバグを発見する．

しかしながら，現時点で検証のためのユーザーモデルを完全に得ることは，特にモデルグラフが非常に大きい場合は現実的には問題がある．また，本稿ではバグに対してかなりの制限が加えられている．例えば，仕様の記述不足から起こるバグについては現在のところ対応できない．さらに，モデルグラフの作成をデバッグ毎に行なっているために非常に時間がかかる．今後はこれらの点を考慮してより良い環境を構築していきたい．

［謝辞］

日頃から御指導いただいている当所，中村英夫主任研究員，本位田真一研究主務に深く感謝します．

〔参考文献〕

〔1〕 Manna,Z.,Wolper,P.,"Synthesis of Communicating Processes from Temporal Logic Specifications,", ACM Trans. on PLS,Vol. 6,No.1,68-93,1984.

〔2〕 Plaisted,D.,"A Decision Procedure for Combinations of Propositional Temporal Logic and Other Specilized Theories",Journal of Automated Reasoning 2,171-190,1986.

〔3〕 Shapiro,E.Y.,"Algorithmic Program Debugging", MIT Press,1983.

〔4〕 内平,川田,"時制論理に基づく実用的仕様記述言語PTSからの手順の生成",情報処理学会研究会60-3, 1988.