組み込みソフトへの数理的アプローチ

第7回

到達可能性と安全性を検証する
── NuSMVとLTSAによる時相論理式の実験

藤倉 俊幸

1 はじめに

前回（2007年4月号，pp.160-165）では時相論理式の二つのタイプ，CTL（計算木論理）とLTL（線形時相論理）の比較を行った．そして，モデル検査ツールNuSMVを使ってLTLの実験場を構築した．構築したといっても，必要な論理変数をIVARで定義して，後は実験対象の論理式を書くだけである．

CTLでは実験場としてステート・マシンも作らなければならないので面倒であると言った．しかし単にどんなときに論理式が真になるか，偽になるか知りたいだけであれば，比較的簡単に実験場を作成できるので，今回はそれを紹介する．例えばVARによって変数を定義するだけでよい場合もあるし，初期値を指定するだけでよい場合もある．IVARよりも初期値の設定などによって状態空間を絞り込めるので，むしろ便利かもしれない．また，LTSA v3.0のLTL実験場も紹介する．これらにより，「共有変数に同時に書き込まない」や「スタック・オーバフローは起こらない」といったことを検証できるようになる．

時相論理式についてはすでにある程度説明してきたが，どんなことを表現できるのか，どんな使い方をするのかを理解しないと実験場を作っても意味がない．また，モデル検査で何ができるのかも見えてこない．実験場に取りかかる前に，まず使い方のパターンを調べる必要がある．

2 モデル検査で何を検査するのか

モデル検査では，検査対象をステート・マシンで表現する．そして，そのステート・マシンの「到達可能性（reachability[注1]）」，「安全性（safety）」，「活性[注2]（liveness）」，「公平性（fairness）」と呼ばれる性質が検査される．

到達可能性は，ある特定の状態に本当に到達することを示そうということである．安全性は，ある条件の下で何かが決して起こらないことを示すこと，活性は，逆にある条件の下で最終的には何かが起こるのを示すこと，公平性は，ある条件の下で何かが何度も起こるのを示すことである．

「動き」についてはこれらの性質が知られているので，動きの仕様を表現する場合にはこれらの性質に言及する必要がある．しかし，実際の仕様書ではほとんどこれらの性質について記述されていないことが多い．いろいろな開発プロセスの仕様書のテンプレートを見ても，機能や性能の項目はあっても動きの項目はない．これらの性質は動きを記述するための座標軸のようなものである．安全性と活性を最初に区別したのはパン屋の問題[注3]で有名なランポート（L. Lamport）らしい．つまり，それまでは座標軸が分離されていなかったわけで，それだけ記述が曖昧（あいまい）だったわけだ．――で，多くの現在使用されている仕様書はランポート以前の状況にある．

とはいっても，到達可能性，安全性，活性，公平性は，ステート・マシンを中心にして考えた性質である．だから，「ある条件で何かが起こらない」から安全だと言われても何のことだかわからない．実際の問題，つまり要求仕様書や設計仕様書に書いてあることは，これらの性質とはほとんど関係ないことばかりである．このことをまず認識してから到達可能性，安全性，活性，公平性の基本的な性質を理解し，仕様書に記述された動きに対して，何が安全性を，何が活性を満たさなければならないか見直す視点を持つ．そうすれば，仕様の抜けを減らすのに役立ち，何を検証すればよいのかが見えてくる．

すべての状態に到達するという型の到達可能性の使い方とデッドロックは起こらないという型の安全性は特に何もしなくてもモデル検査ツールが調べてくれるので楽だが，これだけではモデルを作るかいがないので，もっといろいろ調べたい．そのためには，何が調べられるのかを知らなければ始まらない．

モデル検査を行うときは，何をするか考えて，モデルを作って，そして検査する．何をするかを考えるときに，どのような検査を行うかを明確にすることは原則として忘れてはならない．そして，その際にどのタイプの性質を検査するかまで考えることは，モデルを作ったり，状態爆発を回避するためにモデルを小さくしたりする際などにも重要である．

注1：可能世界の到達可能関係はaccessibility relation.
注2：生存性と訳されることもある．
注3：インターネット上で「ランポートのパン屋」で検索すると見つかる．

KEYWORD――モデル検査，NuSMV，LTSA，安全性，活性，公平性

● 到達可能性

到達可能性の例としては,

- $n < 0$ になりうる
- クリティカル・セクションに入れる
- $n < 0$ にはならない
- クラッシュ状態にはならない

などがある．比較的単純であり使いやすい．形としては,

- ○○になる/ならない
- ××のときに○○になる/ならない

になることが多い．CTL 式では,

EF φ

になる．EF は，現在の状態からいずれφが成立する状態へのパスが存在するという意味であった．E は「存在する」，F は「いずれ」に対応する．φは普通の論理式である．日本語で対応させれば，φは現在形で書いた真偽が決まる文章だと思えばよい．あるいは，もっと直接的にはプログラム言語の if 文などに書く論理式か関係式である．NuSMV で使う CTL 式記号を**図 1** に示す.

ここで「$n < 0$ になり得る」は,

EF（$n < 0$）

となる．「$n < 0$ にはならない」は,

! EF（$n < 0$）

となる．ここで，!は否定(not)を表す．ちなみに,

EF !（$n < 0$）

は，「$n \geqq 0$ になりうる」という意味である．現在の状態からではなく，すべての状態からの到達可能性を CTL 式で表すと,

AG(EF φ)

となる．AG は「すべてのパスのすべての状態で」という意味である．A が「すべての」，G が「いつでも」に対応している．A と E は，パス(実行経路)に対して，「すべて」と「少なくとも一つ存在する」ことを表す．G と F はパス上の状態に対して「すべて」と「少なくとも一つ存在する」ことを表す．このような検査を行うための実験場の例を**図 2** に，実験結果を**図 3** に示す.

一般に A という命題が真なら not A は偽になる．A と not A が両方とも真になることはない．これを排中律と呼ぶ．排中律

```
! 論理式                -- logical not
論理式 & 論理式         -- logical and
論理式 | 論理式         -- logical or
論理式 xor 論理式       -- logical exclusive or
論理式 -> 論理式        -- logical implies
論理式 <-> 論理式       -- logical equivalence
EG 論理式               -- exists globally
EX 論理式               -- exists next state
EF 論理式               -- exists finally
AG 論理式               -- forall globally
AX 論理式               -- forall next state
AF 論理式               -- forall finally
E [ 論理式 U 論理式 ]        -- exists until
A [ 論理式 U 論理式 ]        -- forall until
```

図 1　NuSMV の CTL 式記号

が成立しないと大変なことになる．ところが**図 3** を見ると，$n < 0$ と !($n < 0$) が両方とも真になっている．これらは EF が付いているから，互いに相手の否定になっているわけではないので大丈夫である．否定の！が EF の中に付いていることを見落としてはいけない．互いに否定の関係にするには全体に対して！を付けなければならない．つまり EF の前に付ける必要がある．排中律違反がないとしても「$n < 0$ になりうる」と「$n \geqq 0$ になりうる」が同時に真になるのは，やっぱりおかしい．これはどういうことかというと，EF φは最終的にφが成立するパスが少なくとも一つ存在するという意味で，「$n < 0$ になりうる」場合のパスと「$n \geqq 0$ になりうる」場合のパスは別のものだと解釈すればよいのである．パス(実行経路)が変われば，n の値も変わる．正になることもあるし，負になることもある．ただ，それだけのことを言っている.

つねに初期状態に戻れるという性質は，初期状態を init で表すと,

AG（ EF init）

になる．ちなみにこの性質は LTL 式では表現できないことを前回説明した.

「○○になるまで××である」という条件付きの到達可能性は，E と U の組み合わせで表現できる(**図 4**).

E（$n \neq 0$）U φ

これは，φが真になるまで $n \neq 0$ となるパスが存在するとい

```
MODULE main()
VAR
  n : {-2, -1, 0, 1, 2};

ASSIGN
  init(n) := {-2, -1, 0, 1, 2};
  next(n) := {-2, -1, 0, 1, 2};

SPEC (EF (n < 0))          -- 初期状態から、n < 0 になり得る
SPEC (EF !(n < 0))         -- 初期状態から、n≧0 になり得る
SPEC !(EF (n < 0))         -- 初期状態から、n < 0 にはならない
SPEC AG (EF (n < 0))       -- 何処からでも、n < 0 になり得る
SPEC ! AG (EF (n < 0))     -- 何処からも、n < 0 になることはない
SPEC AG !(EF (n < 0))      -- 何処からでも、n < 0 にはならない
SPEC ! AG !(EF (n < 0))    -- 何処からも、n < 0 にはならないことはない
```

テスト用のモデルnは
−2，−1，0，1，2の範囲を自由に動く

検査式

図 2
実験場-1