モデル検査器NuSMVを利用したテストケース自動生成

門野 雅弥[†] 土屋 達弘[††] 菊野 亨[††]

† 大阪大学 基礎工学部
†† 大阪大学 大学院情報科学研究科
〒565-0871 大阪府吹田市山田丘 1-5

あらまし ソフトウェアの信頼性を向上させるための様々なテスト手法が存在する．本研究では，テスト手法の１つである状態遷移テストに着目する．状態遷移テストは，ソフトウェアシステムの動作を状態図や状態遷移表で表したモデルに基づいて，すべての状態や遷移を網羅するようなテストを行う技術である．この手法を利用して手作業でテストケースを生成できるのは，状態数が非常に少なく遷移条件が単純な場合に限られるため，自動化手法が強く求められている．そこで本研究では，状態図からすべての状態を網羅するテストセットを，モデル検査器 NuSMV を利用して自動生成する手法を提案する．NuSMV の網羅的かつ強力な状態空間探索技術を用いることで，複雑なシステムに対しても漏れなくテストケースを生成することができる．

キーワード ソフトウェアテスト，状態遷移テスト，モデル検査，NuSMV

Automatically Generating Testcases with the NuSMV Model Checker

Masaya KADONO[†], Tatsuhiro TSUCHIYA[††], and Tohru KIKUNO[††]

† School of Engineering Science
†† Graduate School of Information Science Technology, Osaka University
1–5 Yamadaoka, Suita, Osaka 565–0871, Japan

Abstract There are various testing methods of improving the reliability of software. In this study, we consider state transition testing. State transition testing is one of the testing methods based on specifications described as statecharts or state transition tables. This testing method requires that testcases cover all states or all transitions. Manual testcase generation is feasible only when the number of system states is very small and transition conditions are very simple; thus an automatic method is required. In this study, we propose a method that automatically generates testcases with the NuSMV model checker to cover all states. Using the exhaustive state space search method of the NuSMV model checker, we can generate testcases for complex systems in reasonable time.

Key words Software testing, State transition testing, Model checking, NuSMV

1. まえがき

ソフトウェアテストは，プログラムが正しく動作するかどうかを確認する作業であり，ソフトウェアの信頼性を向上させるために必要不可欠な過程である．このとき，できるだけ多くの不具合を発見することが求められる．テストを行う際には，テストケースとそれに対して得られるべき出力を作成しておく必要があり，テストケース設計には様々な手法が用いられる．これらの手法の中でも，仕様に基づく設計手法が注目されている．本研究では，そのような手法の１つである状態遷移テスト [1] に着目し，そのためのテストケース生成について議論する．

状態遷移テストとは，ソフトウェアシステムの動作を状態図 [2] や状態遷移表で表したモデルに基づいて，テストを行う技術である．状態遷移テストでは，状態を遷移させるきっかけとなるイベントの列をテストケースとする．テストケース設計基準として，すべての状態の網羅やすべての遷移の網羅が用いられる．本研究では前者の基準を用い，状態図上のすべての状態を網羅するテストセットを生成することを考える．

一般にソフトウェアシステムは複数の変数を持ち，それらの値の組み合わせがシステム状態を表わす．よって，変数とその取りうる値が増えるほどシステム状態数も多くなる．このようなシステム状態を考慮して状態図上のすべての状態を網羅するテストケースを設計するのは，非常に手間がかかり，現実的には不可能である．そこで本研究では，モデル検査器 NuSMV [3] を利用してテストケースを自動生成する手法を提案する．具体的には，専用言語で記述された状態図を入力として，状態図の

各状態に到る経路をNuSMVの反例から得て，それらをテストケースとして利用する．また，テストセットが既に存在する場合には，それを初期テストセットとして入力できるものとし，初期テストセットが網羅できない各状態への経路をNuSMVの反例から得て，それらを新たなテストケースとして初期テストセットに加えることも可能にする．なお，モデル検査を利用してテスト生成を行う研究には，[4]〜[6]がある．

以降，2.節ではモデル検査について述べる．3.節では本研究で提案する，モデル検査器NuSMVを用いたテストケースの生成について述べる．NuSMVを利用するためには，対象のソフトウェアの動作をSMVコードで記述する必要がある．本研究では，状態図からNuSMVのSMVコードへの変換はモデル検査支援ソフトウェア[7]を利用する．4.節では実装したツールを適用した結果について述べる．最後に，5.節では本研究のまとめと今後の課題について述べる．

2. モデル検査

本節では，モデル検査について述べる．まず概要を述べ，続いて本研究で利用したモデル検査器NuSMVについて述べる．

2.1 概　　要

モデル検査とは，有限状態遷移システムの全状態を網羅的に探索して，特定の性質を満たしているかどうかを検証する技術である．有限状態遷移システムとは，状態の集合，初期状態の集合，状態間の遷移関係の3要素から成る．また，各状態に対して必ずその状態を遷移元とする遷移が存在するものとする．検証項目はすべての初期状態に対して満たされているかどうか評価される．満たされていない場合には，反例が出力される．反例とは，検証項目が満たされないことを示す状態系列である．

2.2 モデル検査器NuSMV

NuSMVは，記号モデル検査（Symbolic Model Checking）[8]ツールの1つである．記号モデル検査は，状態集合や状態間の遷移を数式で記号的に表現し，それらの数式上の処理によって状態探索を実現し，処理の高速化を実現する手法である．本研究でNuSMVを利用するのは，以下のような特徴があるためである．

- 状態空間の網羅的な自動探索技術が利用できる．
- 状態図からSMVコードを生成する研究が多い[9][10]．

NuSMVは，LTL（線形時相論理）とCTL（計算木論理）で記述した性質を検証できる．これらは時相論理の一種である．本研究ではCTLを用いる．本研究で用いた時相作用素を以下にまとめる．

- AG p：pがすべての経路で常に成り立つ
- AX p：pがすべての次状態で成り立つ
- EF p：pがある経路でいずれ成り立つ

3. 提案手法

本節では，本研究で提案・実装するモデル検査器NuSMVを利用したテストセット自動生成手法について述べる．まずテストセット生成手順の概要を述べ，続いてその詳細について述べる．

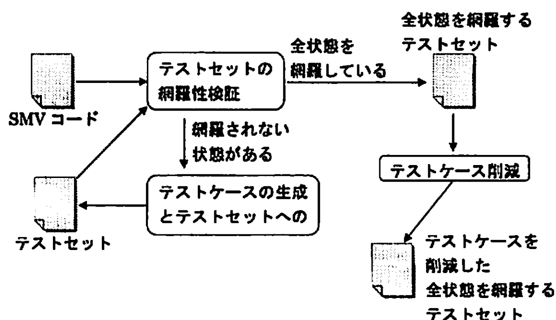

図1 テストセット生成手順

3.1 テストセット生成手順

本研究で提案するテストセット生成の手順を図1に示す．

入力として状態図の情報を含むSMVコードとテストセットを考える．まず，SMVコードとテストセットから，テストセットが全状態を網羅しているかを検証する．テストセットが全状態を網羅しない場合は，網羅できなかった状態に到達できるテストケースを生成してテストセットに追加する．網羅性の検証とテストケースの生成・追加を，テストセットが全状態を網羅できるようになるまで繰り返し，最終的に全状態を網羅できるテストセットを出力する．

1.節で述べたように，初期テストセットが与えられる場合とそうではない場合がある．図1の手順において，初期テストセットが入力として与えられる場合は，そのテストセットが全状態を網羅しているかを検証すればよい．テストセットが入力として与えられない場合は，長さ0のテストケースを1つ含むテストセットが与えられたものとする．この場合でも初期状態には必ず到達できるので，初期状態のみを網羅するテストセットが与えられたと考えることができ，初期テストセットが与えられた場合と同様の手順でテストセットを生成することができる．

3.2 テストセットの網羅性検証

テストセットが全状態を網羅しているかという網羅性の検証について，例を用いて説明する．図2の状態図が入力モデルとして与えられる場合を考える．このとき，図3のようなモデル（網羅性検証用モデル）をSMVコードで記述し，それに対してモデル検査器NuSMVを用いて網羅性の検証を行える．

状態図の見方について説明する．図3では角の丸い四角形が状態を表しており，黒塗りの円から出た矢印が初期状態を指し示している．また，状態間を結ぶ矢印は遷移を表しており，以下のラベルが付されている．

- イベント[ガード条件]/アクション

イベントは状態を遷移させるきっかけである．ガード条件は真偽値を値にもち，その値が真であるときのみ遷移が許される条件である．また，アクションは状態遷移が起きたときに実行される動作であり，図3では，どのイベントを発生させるかを記述している．

状態A内は破線で区切られ，状態Bと状態Cが状態Aの中

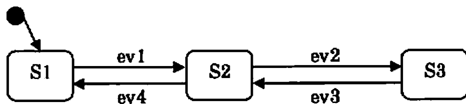

図 2　入力モデル

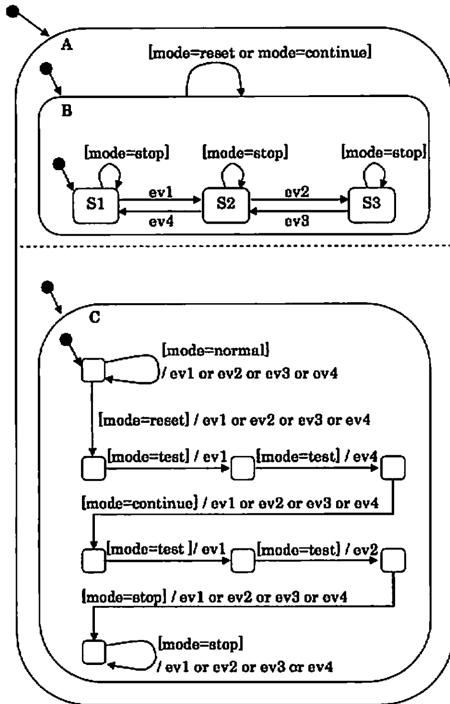

図 3　網羅性検証用モデル

に入れ子に描かれ (階層化)，さらに状態 B と状態 C の中に状態図が描かれている．これは状態 B と状態 C の中の状態が並行して遷移することを意味している．

3.2.1　網羅性検証用モデル

網羅性検証用モデルの例を図 3 に示す．このモデルの例では，入力される状態図（入力モデル）は図 2 であり，以下のテストセットが既にあるものとしている．

- {(ev1, ev4), (ev1, ev2)}

網羅性検証用モデルは，新たに導入した変数 mode を用いて，その値によって入力モデルの遷移の方法と，イベントの発生方法を変化させるモデルである．変数 mode の値は，{normal, reset, test, continue, stop} のうちのいずれかであり，初期値は normal である．以下に，mode の値に対する入力モデルの遷移の方法と mode の値の変化を示す．

- mode = normal (初期値)

– イベントをランダムに発生させ，入力モデルを初期状態から状態遷移させる．任意のタイミングで mode の値を reset にする．

- mode = reset

– 入力モデルの状態を初期状態に戻す．テストセットが存在しない場合には mode の値を stop にし，そうでなければ mode の値を test にする．

- mode = test

– テストセットに含まれる 1 つのテストケースを使ってイベントを順に発生させ，入力モデルの状態を遷移させる．テストケースによる状態遷移が終了した後，テストセットにまだ使っていないテストケースがあれば mode を continue にする．すべてのテストケースを使っていれば mode の値を stop にする．

- mode = continue

– 入力モデルの状態を初期状態に戻し，mode の値を test にする．

- mode = stop

– 入力モデルの状態をそれ以上遷移させず，mode の値も stop のままとする．

図 3 を用いて，網羅性検証用モデルについて具体的に説明する．

状態 B 内の遷移は，入力モデルの状態遷移を表わしている．mode=normal と mode=test のときはイベントに従って状態が遷移する．mode=reset あるいは mode=continue のときは，状態 B は自己遷移し，状態 B 内のどの状態にあっても初期状態 S1 に遷移することを表している．mode=stop のときは，後述するように，状態 C 内でイベントが発生しなくなるので，それ以上状態遷移は起こらない．

状態 C 内の遷移は，イベントをどのように発生させるかを表している．mode=normal のときにはランダムにイベントを発生させている．mode=reset となると状態を遷移させ，テストセットの 1 つ目のテストケース (ev1, ev4) に従ってイベントを順次発生させる．テストケースのイベントをすべて発生させ終えると mode=continue となり，2 つ目のテストケース (ev1, ev2) に従ってイベントを順次発生させる．テストセットにはこれ以上テストケースが存在しないので，mode=stop となり停止し，イベントは発生させない．

以上のようなモデルを考え，そのモデルに対してモデル検査を行うことでテストセットが全状態を網羅しているかを検証する．

3.2.2　検 証 項 目

3.2.1 節で述べたモデルに対して，以下の検証項目を検証することでテストセットがすべての状態を網羅しているかを調べることができる．

- AG ((mode = reset) -> AX EF(pc = r_pc))

変数 mode については既に述べた．pc は状態名を表す変数である（本研究では入力となる SMV コードにおいて，状態名を表す変数は pc であるとしている）．r_pc は変数 mode の値が reset になったときの状態名を保持する変数である．

検証項目の表す性質について述べる．

上記の検証項目は，「初期状態から到達可能なすべての状態において，mode の値が reset であれば，そのすべての次状態において，そこから始まるパスで，いつかは pc の値が r_pc に等しくなるものが存在する」と言い表せる．つまり，mode の値が reset になったときの状態名を r_pc に保持しておき，その後，

テストセットに含まれる各テストケースのイベントに従って遷移させたときに到達する状態が，いつかは r_pc に等しくなるという性質を意味する．mode の値が reset となるのは任意のタイミングであるので，イベントをランダムに発生させたときに到達できるすべての状態に対して，テストセットを使って到達できるかどうかを検証することができる．

この性質を網羅性検証用モデルに対して検証したときに，成り立っていればそのテストセットがすべての状態を網羅するテストセットであることがわかる．成り立っていなければ NuSMV より反例が出力される．この反例から新たなテストケースを生成する．その方法について次節で述べる．

3.3 テストケースの生成

先の検証項目に対して得られる反例は，テストセットを使って，ある 1 つの状態には到達できないということを示す状態系列である．つまり，得られた反例において，変数 mode の値が reset となるまでの状態系列の各状態から，イベントを表わす変数の値のみを取り出すことで，そのイベントの列がテストケースとして利用できる．また，mode の値が reset となる状態における変数の値を取り出すことで，テストケースによって，各変数の値が最終的にどのような値になるかという情報も得ることができる．さらに，そのテストケースで到達できるすべての状態名も反例から得られる．

以上のようにして得られたテストケースを新たにテストセットに追加する．

3.4 テストケースの削減

生成されたテストセットには，除去してもテストセットが全状態を網羅できることには変わりがないテストケースが存在することがある．本研究では，そのようなテストセットから除去可能なテストケースを見つける方法を 2 つ提案する．

1 つ目は，あるテストケースの prefix（前方一致）となっているテストケースは，除去する方法である．例えば，テストケース (ev1,ev2) と (ev1,ev2,ev3) がある場合，前者は後者の prefix であるので除去可能である．

2 つ目は，網羅する状態の集合が，他のテストケースが網羅する状態の集合の部分集合となっているテストケースは，除去する方法である．例えば，網羅する状態の集合が {S1,S2} であるテストケースと，網羅する状態の集合が {S1,S2,S3} であるテストケースがある場合，前者のテストケースの状態集合を網羅するテストケースは除去可能である．

4. 適用実験

本節では，提案手法を実装したツールを，ストップウォッチ [11] と TCAS II（空中衝突防止装置）の仕様の一部 [12] に適用した結果を示す．以降，テストケースの prefix に着目して削減する方法を削減手法 1，網羅する状態の集合に着目して削減する方法を削減手法 2 と表記する．適用結果は，削減手法を利用せずに実行した場合と，利用して実行した場合の実行時間，生成されるテストケースの数，各テストケースの長さを掲載する．

実行環境は以下の通りである．

表 1 ストップウォッチの持つ変数

変数名	変数の取りうる値
pc	Reset, Running, Lap_stop, Lap
ev	LAP, TIC, START
cent	0〜99
disp_cent	0〜99
sec	0〜59
disp_sec	0〜59
min	0〜4
disp_min	0〜4

表 2 ストップウォッチ:
状態図 A･1 の全 4 状態に適用した結果

削減手法	実行時間	テストケース数	各テストケース長
削減手法 1	7 分 1 秒	1	4
削減手法 2	7 分 1 秒	1	4
無し	7 分 0 秒	3	2, 3, 4

表 3 TCAS II の持つ変数

変数名	変数の取りうる値
ev	evu, evv, up, down, test, add1, add10, sub1, sub10
u	0, 1
v	0, 1
w	0, 1
switch	up, down, test
alt	0〜200
prev-alt	0〜200
Alt-Layer	High, Mid, Low
Alarm	Shutdown, Operating
Mode	Off, On
Volume	1, 2
time-Mid	0〜5

- CPU : Core 2 Duo U7500 1.06GHz
- RAM : 1GB
- OS : Windows Vista

4.1 ストップウォッチ

ストップウォッチは，表 1 の変数を持つ．状態図は付録の図 A･1 に掲載する．なお，NuSMV が探索するストップウォッチの到達可能なシステム状態数は，1.44×10^8 であった．この状態数は，ストップウォッチが持つ変数が取りうる組み合わせの総数である．これは，NuSMV の"-r"オプションを用いて得られた．

この実験で考えるのは，状態図 A･1 における Reset，Running，Lap，Lap_stop の 4 状態の網羅である．

適用結果を表 2 にまとめる．

4.2 TCAS II

TCAS II は，表 3 の変数を持つ．状態図は付録の図 A･2 に掲載する．なお，NuSMV が探索する TCAS II の到達可能なシステム状態数は，1.42×10^5 であった．

TCAS II は，状態 Alt-Layer，Alarm，Mode，Volume を持ち，それぞれの状態内にいくつかの階層化された状態がある．

この実験では，以下の4つの状態網羅を考える．

- 状態図 A-2 の状態 Alt-Layer における High, Mid, Low の全3状態の網羅
- 状態図 A-2 の状態 Alarm における Shutdown, Operating の全2状態の網羅
- 状態図 A-2 の状態 Mode における On, Off の全2状態の網羅
- 状態図 A-2 の状態 Volume における 1, 2 の全2状態の網羅

適用結果を表4〜7にそれぞれまとめる．

4.3 考察

ストップウォッチと TCAS II に適用した結果，状態図におけるすべての状態をテストケースが生成されることを確認した．

削減手法を利用することで，ストップウォッチのテストケースの数が削減されることを確認した．TCAS II では，テストケースはそれ以上削減されなかった．

前述したとおり，ストップウォッチの到達可能なシステム状態数は 1.44×10^8，TCAS II の到達可能なシステム状態数は 1.42×10^5 であり，実験結果からシステム状態数の多い前者の方が，実行時間がかかることが分かる．これは，システム状態数が多くなるほど，NuSMV の探索する状態数も大きくなり，それだけ探索に要する時間がかかると考えられるためである．

また，表4の実行時間と，表5〜7の実行時間を比較すると，実行時間は生成されるテストケースの数に依存すると予測される．これは，テストケースの数が多いほど，NuSMV による検証が必要になると考えられるためである．例えば n 個のテストケースが生成される場合，網羅性の検証とテストケース生成を行うために，NuSMV を利用して n 回検証を行うことになる．よって，テストケースの数 n が大きくなるほど時間がかかる．

削減手法を使う場合と使わない場合の実行時間の差は，今回行った実験ではほとんどなかった．これは，生成されるテストケースの数が少なかったためであると考えられる．

表4 TCAS II:
状態図 A-2 の状態 Alt-Layer 内の全3状態に対する適用結果

削減手法	実行時間	テストケース数	各テストケース長
削減手法1	58秒	2	5, 11
削減手法2	58秒	2	5, 11
無し	59秒	2	5, 11

表5 TCAS II:
状態図 A-2 の状態 Alarm 内の全2状態に対する適用結果

削減手法	実行時間	テストケース数	各テストケース長
削減手法1	28秒	1	3
削減手法2	28秒	1	3
無し	28秒	1	3

表6 TCAS II:
状態図 A-2 の状態 Mode 内の全2状態に対する適用結果

削減手法	実行時間	テストケース数	各テストケース長
削減手法2	34秒	1	4
削減手法2	34秒	1	4
無し	34秒	1	4

表7 TCAS II:
状態図 A-2 の状態 Volume 内の全2状態に対する適用結果

削減手法	実行時間	テストケース数	各テストケース長
削減手法1	30秒	1	6
削減手法2	30秒	1	6
無し	30秒	1	6

5. まとめと今後の課題

本研究では，モデル検査器 NuSMV を利用して，状態図からテストケースを自動生成する手法の提案を行った．具体的には，テストセットが全状態を網羅しているかを検証し，網羅できていない状態に到達する経路を NuSMV の反例から得て，反例からイベントを表す変数の値を取り出してテストケースとする．また，生成されたテストケースの prefix に着目してテストケースを削減する方法と，テストケースが網羅する状態の集合に着目してテストケースを削減する方法についても提案した．

提案手法を実装し，ケーススタディとしてストップウォッチと TCAS II の2つのシステムに適用した．この実験により，システム状態数が増えるにつれて実行時間が大きくなることが分かったが，適用した例では現実的な範囲に収まった．また，人手によるテスト設計と比較すると，テストケースが漏れなく自動生成できるのは，大きな利点であると考えられる．

提案した手法では，複数の並行動作する状態遷移は扱えない．TCAS II に対する適用実験では，並行動作する各状態遷移に対し別々に実験を行ったが，システム全体の状態は並行する状態の組み合わせであるため，それらの網羅にも対応できることが望ましい．複数の並行する状態を扱えるようにし，それらの状態の取りうるすべての組み合わせを網羅するようなテストケースを生成できるように，本研究での提案手法を拡張することが今後の課題である．また，本研究では手法の実装と適用実験を行ったが，生成されたテストケースが実際のシステムのテストに有用かどうかも確認する必要がある．この点についても，今後の課題である．

文献

[1] Lee Copeland, 宗雅彦："はじめて学ぶソフトウェアのテスト技法", 第7章, 日経 BP 社 (2005).

[2] D. Harel: "Statecharts: A visual formalism for complex systems", Sci. Comput. Program., 8, 3, pp. 231–274 (1987).

[3] "NuSMV home page", http://nusmv.irst.itc.it/.

[4] A.Gargantini and C.L.Heitmeyer: "Using model checking to generate tests from requirements specifications", European Software Engineering Conference, pp. 146–162 (1999).

[5] P. Ammann, P. Black and W.Majurski: "Using model checking to generate tests from specifications", the Second IEEE International Conference on Formal Engineering Methods, pp. 46–54 (1998).

[6] B. Lindstom, P. Pettersson and J. Offutt: "Generating trace-sets for model-based testing", 18th IEEE International Symposium on Software Reliability Engineering, pp. 171–180 (2007).

[7] "実践!ソフトウェアモデル検査", http://modelcheck.jp/index.html.

[8] K. L. McMillan: "Symbolic Model Checking", Kluwer Academic (1993).

[9] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin and J. D. Reese: "Model checking large software specifications", IEEE Trans. Software Eng., **24**, 7, pp. 498–520 (1998).
[10] W. Chan, R. J. Anderson, P. Beame, D. H. Jones, D. Notkin and W. E. Warner: "Optimizing symbolic model checking for statecharts", IEEE Trans. Software Eng., **27**, 2, pp. 170–190 (2001).
[11] Gregoire Hamon,Leonardo de Moura and John Rushby: "Automated Test Generation with SAL", Technical report, SRI International (2005).
[12] William Chan: "Symbolic Model Checking for Large Software Specifications", Ph.D. dissertation, University of Washington (1999).

付　　録

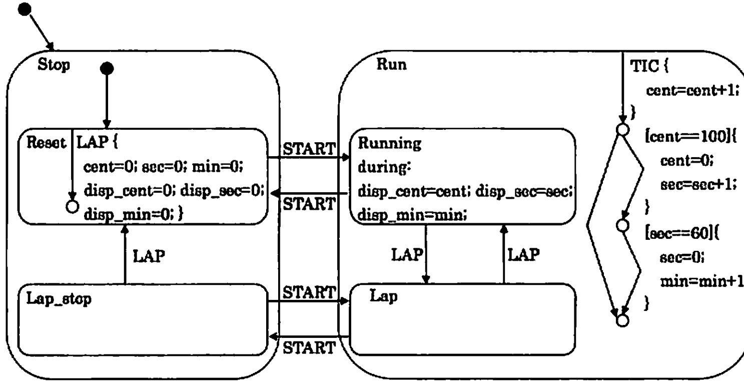

図 A·1　ストップウォッチの状態図

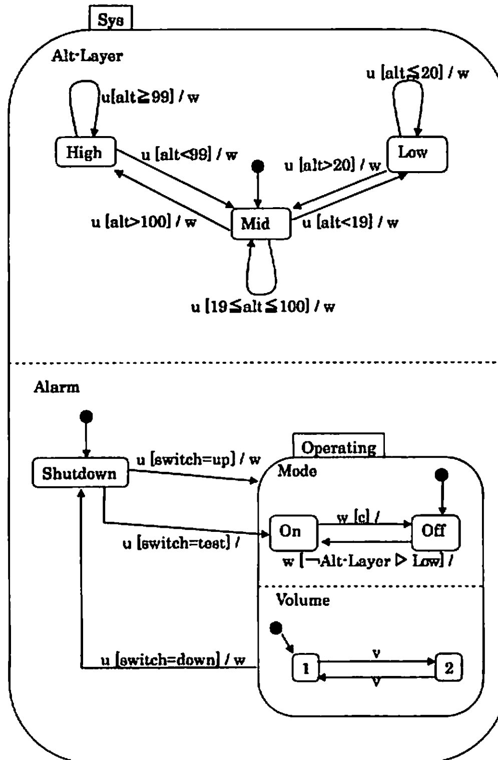

図 A·2　TCAS II の状態図