ソフトウェア工学 66－1
（1989. 6. 27）

システム設計における仕様検証の一手法

手島 文彰[*1] 井上 勝博[*1] 岸本 卓也[*2] 三原 幸博[*1]

[*1]システム・ソフトウェア技術研究所 [*2]家電技術研究所
株式会社 東芝

形式的仕様記述に基づく動的検証手法について述べる。オートマトンの概念に基づいて、有限状態機械モデルによって表現されたシステムの設計仕様に対して、実現されたシステムの動作が正しいかどうかを客観的に判定することが可能になった。そして、システムや検査機器の誤差が検証にどのような影響を与えるかについて考察し、それらを解決するためにオートマトンを拡張した。さらに、この手法を実際の製品のシステムテストに適用した結果、従来の方法では検出することが難しかった誤りを検出することができた。また、この手法により仕様の完全性を調べることが可能であることがわかった。

A METHOD OF SPECIFICATION VERIFICATION IN SYSTEM DESIGN

Fumiaki TESHIMA [*1] Katsuhiro INOUE [*1] Takuya KISHIMOTO [*2] Yukihiro MIHARA[*1]

[*1]Systems and Software Engineering Laboratory
70 Yanagi-cho, Saiwai-ku, Kawasaki,
210 JAPAN

[*2]Consumer Products Engineering Laboratory
8 Shinsugita-cho, Isogo-ku, Yokohama,
235 JAPAN

TOSHIBA Corporation

ABSTRACT

Dynamic verification method based on formal specification is discussed. Using the Finite State Transition Model as a specification description model, system design specification is modeled. And it is possible to validate that dynamic system behavior is consistent with its system design specification objectively. For this, how tolerance of system and monitor equipment effect the verification is examined and a method for solving this problem is proposed. When this method was applied to the practical product, faults, which were not to detectable in usual way, had been detected. Using this method,we were able to confirm that system testing can be automated highly.

1　はじめに

システム開発の際に、高品質・高信頼のシステムを作成するための方策として、システム仕様記述への形式的記述の導入が注目されている。これは、システム記述時にシステム仕様に論理的矛盾がないかどうかを検証することができる[1]、あるいはその仕様を直接または間接に実行することによって仕様をより完全なものにできるためである[2]。ここでは、形式的に記述されたときの仕様の完全性や厳密性を用いて、実現されたシステムがその仕様を満たしているかどうかを機械的に調べるために形式的仕様記述に基づく動的検証について、それをシステムテストに適用する観点から考察を加えた。

動的検証とは、実際にシステムを実行させ、そのときのシステムの動作がその仕様と矛盾しないことを確認し、システムと仕様との等価性を保証しようとするものであり、ここでは有限状態機械モデルによりモデル化された仕様に対して、オートマトンの概念を用いた検証手法を提案する。

以下、第２章では仕様記述モデルについて述べた後、本検証手法の基本的概念を示す。第３章では本手法を実際のシステムテストに適用しようとしたときの問題点について述べ、第４章でそれを解決するためにオートマトンを拡張する。第５章には本手法を実現した検証システムの構成を示し、第６章では本手法の評価および考察を行なう。そして、本手法による効果を第７章で述べ、第８章では本手法の適用可能な分野について考察する。最後に、今後の課題について述べる。

2　検証手法

本章では、今回提案する検証手法の概要を述べる。まず、仕様を形式的に記述するための仕様記述モデルを示し、その特徴を述べる。次に、本手法の基本的概念について述べる。

2．1　仕様記述モデル

本手法では、有限状態機械モデル(Finite State Machine Model)によりシステムの設計仕様をモデル化する。有限状態機械モデルは、システムのそれまでの入力イベントと出力動作の履歴を状態に対応させ、新たな入力イベントにより、そのときの状態から定まる動作および状態遷移をグラフを用いて図的に表現するものである。すなわち、各状態（ノード）をその入力イベントと出力動作（アクション）を付けた矢印（アーク）で接続することにより、外界や内部からの刺激（イベント）に継続的に対応するシステムの振舞いを視覚的に表現できるという利点をもつ。逆に、システムの機能が複雑化するにつれて、状態数が指数関数的に増大し、状態遷移図の規模が膨大となり、システム全体の理解が難しいなどの問題がある。

2．2　基本的概念

ここでは、本手法の基本的概念について述べる。この手法は、オートマトンの概念に基づいている。すなわち、有限状態機械モデルに基づく形式言語でシステムの設計仕様を記述し、その設計仕様を仮想的に実現したオートマトンに対して、実際のシステムの動作をモニタしたデータを入力系列として与えたとき、それが受理されるか否かにより検証を行う。これにより、システムの動作と仕様との等価性をいうことができる。図１は、このことを概念的に示したものである。

ただし、ここでいう等価とは、設計仕様に対してであり、設計者の意図したものに対してではない。つまり、そのような仕様は設計者の意図が正しく表現されていることを前提としている。そして、設計仕様のすべて（システムの入力空間全体）に対して上記の等価性がいえたとき、システムの論理的な正しさが証明されたことになる。

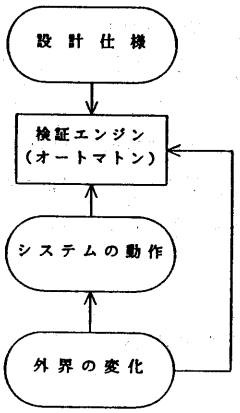

図1　検証手法の基本的概念

本システムにおける検証項目は、以下の通りである。

（1）部分的な正しさ

部分的な正しさとは、与えられた入力系列のある時区間において、発生した任意の事象およびそれらの事象間の関係が与えられた仕様と矛盾しないことをいう。部分的な正しさでは、制御シーケンスおよび事象発生のタイミングなどの無矛盾がいえる。

（2）全体的な正しさ

全体的な正しさとは、与えられた入力系列に対して、発生したすべての事象およびそれらの関係が与えられた仕様と矛盾しないことをいう。全体的な正しさでは、異常な事象が存在しないことがいえる。

3　実システムへの適用

本手法の有効性を評価するために、この手法を実際のシステムのテストに適用した。今回対象としたシステムは、マイコン組み込みの家庭用冷蔵庫である。これは、仕様が有限状態機械モデルによりモデル化し易いこと、仕様がそれほど大きくないことなどの理由から選択した。

本章では、まず対象となったシステムの仕様記述例を示した後、本手法を適用しようとしたときの問題点を整理する。

3．1　仕様記述例

図2に、冷蔵庫の制御系のシステム設計の仕様の一部を有限状態機械モデルを用いてモデル化した例を示す。この仕様では、次のことを表現している。

（1）　冷凍室内の温度をＴγ－Ｔαの間に保つ

（2）　コンプレッサが一度オフしたら、少なくともｔΔ経過するまで、コンプレッサをオンできない

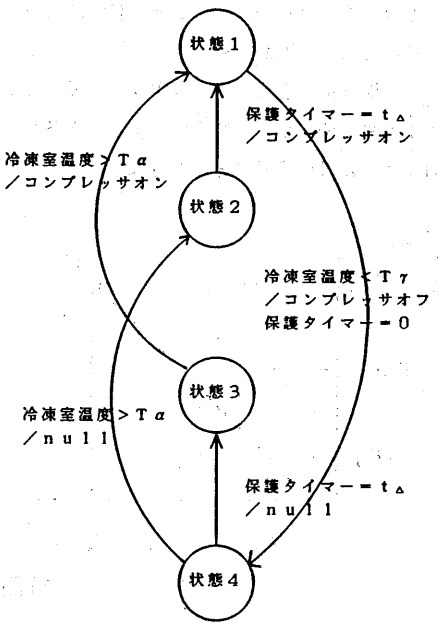

図2．仕様記述例

3．2　問題点

検証を行う上で問題となったことについて述べる。

3．2．1　誤差

まず、検証を行なう上で問題となるのは、さまざまな誤差である。誤差としては、次のようなものが考られる。

・センサー誤差
・サンプリング誤差
・システム誤差

ここで、これらの誤差が検証にどのような影響を及ぼすかについて、その代表的な誤差であるセンサー誤差を例にして述べる。

図2の仕様記述例において、"冷凍室温度＞Tα"というイベントは、冷凍室の温度がTαより高くなったときに発生する。これは、冷凍室の温度を測定しているセンサーの電圧値により識別される。このとき、センサーには性能のばらつきがあり、そのばらつきは±Tβの範囲内である。

このため、"冷凍室温度＞Tα"というイベントは、図3に示されるようなTα－TβからTα＋Tβまでのどこかで発生するはずであるが、これを正確に検出することはできない。

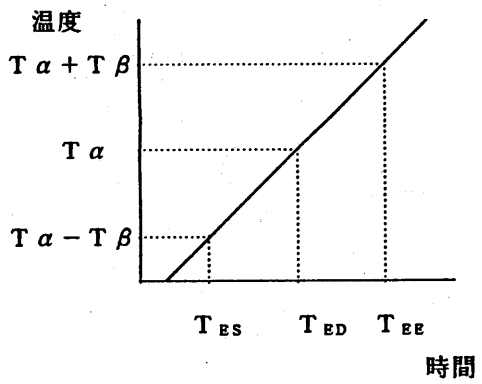

図3　センサー誤差（その1）

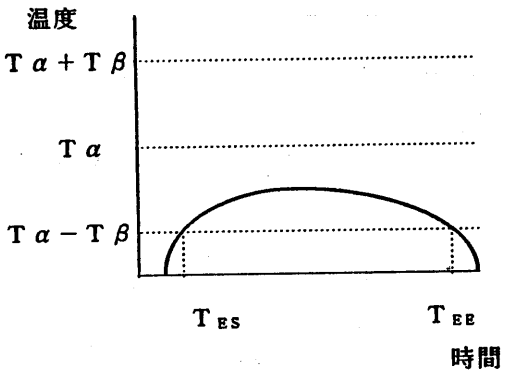

図4　センサー誤差（その2）

また、冷凍室の温度が図4のように変化したときには、"冷凍室温度＞Tα"というイベントが発生したかどうかさえ検出することはできない。これは、あるセンサーでは冷凍室の温度がTαになったと認識し、別なセンサーではまだTαになっていないと認識しているかもしれないからである。

3．2．2　仕様の曖昧さ

また、システム設計では、仕様が厳密に記述されていない場合もある。例えば、次のようなものである。

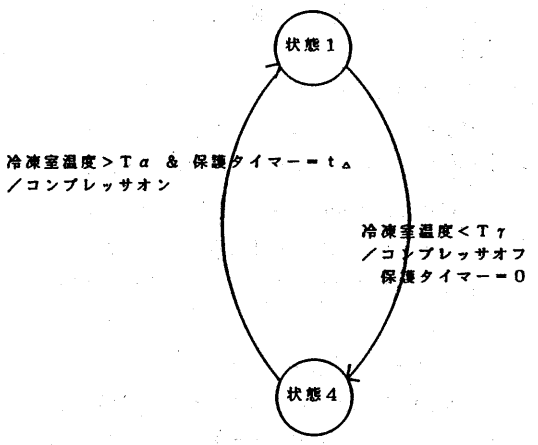

図5　仕様記述例

図5は、設計者にとっては図2の仕様と全く同じことを意味している。しかしながら、図5は状態4から状態1へ遷移するためには、2つのイベントが同時に発生しなければならないという意味に解釈することもできる。つまり、図5には遷移条件においてイベントの発生順序に関する記述が省略されているのである。このような仕様の曖昧さは、仕様の理解性をあげるためには重要であるが、この曖昧さが検証に大きな影響を与えるのである。

3．2．3　内部イベント

また、システムは一般に内部状態を持ち、それに基づいた制御を行う。このような制御の代表的な例としては、内部タイマーを用いた制御などがあげられるが、今回対象としたマイコン組み込みの冷蔵庫のようなシステムでは、そのようなシステムの内部状態をモニタすることは非常に難しい。そのため、システムの内部的なイベントや内部状態を変えるようなアクションを検出することはできない。

内部イベントの例としては、図2の“保護タイマー＝t_{Δ}”がある。また、内部アクションの例としては、図2の“保護タイマー＝0”がある。

また、イベントの中にはある状態のときにある条件が満たされるか否かでイベント発生の有無を決定するものがある。つまり、イベントが発生したかどうかは、システムの内部状態に依存するのである。このようなイベントを特に状態イベントと呼ぶことにするが、システムがいつどのような状態になったのかを正確に知ることができない。このため、このようなイベントの発生を検出することはできない。

状態イベントの例としては、図2の仕様において、状態4に移ったときに冷凍室の温度がT_{α}より高かったならば、状態2に移るというような制御がある。

3．2．4　決定不能

これまで検証において問題となると考えられたことについて述べてきた。ここでは、それらが検証にどのような影響を及ぼすのかについて述べる。

これらの問題は、検証において決定不能と呼ばれる状況を引き起こす原因となる。決定不能とは、発生したイベントやアクションの間で順序関係がつけられないことであり、以下にその代表的な例を示す。

［場合1］

図6で、e_iかe_jの少なくとも1つが検出不可能なイベントであるとき、決定不能になる。

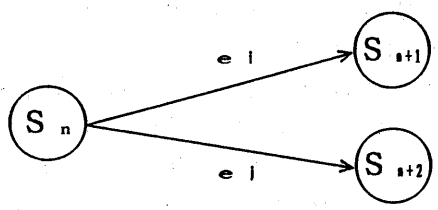

図6　決定不能（その1）

［場合2］

図7で、e_iかe_jのどちらか1つがまだ確定的でないとき、決定不能になる。例えば、e_iが確定的でないということは、e_iはその時刻において発生しているかもしれないし、発生していないかもしれないということを意味する。つまり、もしe_iが本当に発生しているのであれば、e_iとe_jの順序関係は、$e_i \rightarrow e_j$となる。そして、e_iがまだ発生していないのであれば、e_jとなる。

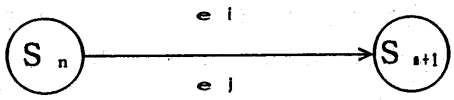

図7　決定不能（その2）

4　解決方法

このような問題を解決するために、次のような概念を導入し、オートマトンを拡張した。

4．1　時区間表現

まず、誤差を陽に記述できるような概念を取り入れることを考えた。このため、まず誤差の範囲を明確するために事象の発生時間を時点ではなく時区間により表現する。すなわち、その事象の発生した可能性を生じた時点、その事象が発生したことが確定した時点、およびその事象が完結した時点により事象の発生時間を表現する。つまり、事象の発生時間は３つの時点の系列により表現される。

この時区間表現では、図３の“冷凍室温度＞Tα”というイベントは、温度がTα－Tβになったとき発生開始、温度がTαになったときに発生確定、そして温度がTα＋Tβになったとき発生終了となる。

このように、誤差範囲を明確にすることは、仕様検証だけでなく、タイミング検証などにおいても非常に重要であると思われる。

4．2　仕様詳細化

また、仕様が時間に関して曖昧である場合には、仕様の詳細化を行って、仕様を厳密にする。これは、図５の仕様を図２に示されるような仕様に等価変換することを意味する。つまり、仕様に冗長性を持たせることによって、仕様の曖昧性を排除している。

4．3　疑似イベント

前章で述べたように、システムの内部状態をモニタすることは難しい。しかしながら、システムは外界の変化に対応して絶えずアクションを起しており、また外界の変化をモニタすることは容易である。このため、それらからシステムの内部状態を推定することは可能である。

そこで、オートマトンに内部状態を持たせ、検証における状態遷移にしたがって内部アクションを自分自身に対して実行することにより、システムと同様な内部状態を仮想的に実現することができる。そして、その内部状態に基づいて疑似的な内部イベントや状態イベントを発生させてやれば、システムの内部状態をモニタできないことによる問題は解決される。

また、状態イベントを他のイベントと同じように扱うことができるようにするために、状態時間という概念を導入する。状態時間とはその状態に移った時間であり、状態に時間の概念を導入することによって、状態イベントに時間情報を付加することができる。

そこで、まず状態時間を定義する。状態時間とは、その状態に遷移した時間であるので、その状態遷移を可能にしたイベントとアクションの発生時間により決定されると考える。イベントやアクションは時区間により表現されているので、状態時間も時区間により表現され、その状態遷移を可能にしたイベントとアクションの発生時間の論理積により決まる。

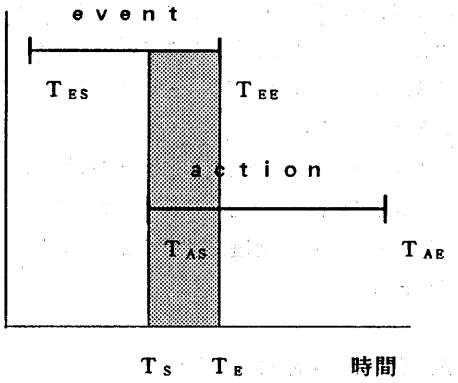

図８　状態時間

図８で、イベントとアクションの発生時間は、それぞれ$T_{ES}-T_{EE}$，$T_{AS}-T_{AE}$であり、状態時間はT_S-T_Eとなる。そして、状態イベントは、その状態の状態時間を発生時間とする。

内部イベントについては、その内部状態がイベント発生条件を満たすかどうかを調べ、もし条件を満たすならばイベントを発生させるようにする。そして、内部イベントについても他のイベントと同様に扱うことができるようにする。

4．4　オートマトンの拡張

前節までに述べた概念により決定不能問題の一つは解決することができる。もう一つの問題は、不確定性の問題である。これを解決するために、非決定性オートマトンの概念を導入して、オートマトンを拡張する。非決定性オートマトンでは、ある入力系列に対して、初期状態から最終状態に至る状態遷移の列でこの入力系列に対応するものがあるとき、その入力系列はオートマトンで受理されたと考える。つまり、全ての可能な状態遷移の中に最終状態に達するものが一つでも存在するか否かを調べるのである。

また、オートマトンにおける拒否(reject)については、次のことを含めて評価する。

（1）事象の属性

本手法では、事象をいくつかのクラスに分類し、それぞれのクラスに対して「確定的」「不確定的」のいずれかの属性を付けている。例えば、次のようなものである。

（a）外部事象

・ディジタル的な事象　‥‥　確定的

（例）　スイッチ

・アナログ的な事象　‥‥‥　不確定的

（例）　温度

（b）内部事象　‥‥‥‥‥‥　確定的

（例）　タイマー

（2）事象の発生状態

評価対象となった事象が、その時点で発生が確定している場合と、まだ確定していない場合がある。そのため、そのような事象の発生状態についても評価を行う。

そして、オートマトンで事象の入力系列が拒否されたときには、次のいずれかに判定する。

・保留‥このとき、次の事象を評価する。
ただし、拒否の原因となった事象は、次の評価においてもその対象となる。

・無視‥このとき、次の事象を評価する。
ただし、拒否の原因となった事象は評価されない。

・矛盾‥この状態遷移の系列をあきらめ、他の状態遷移を調べる。

そして、全ての可能な状態遷移の中に最終状態に到達するものが一つも存在しないときに、その入力系列は拒否されたと定義する。つまり、そのようなシステムの動作は仕様と矛盾しているとみなす。

以上のことにより、検証における問題は基本的には解決することができる。

5　検証システムの構成

図9に、本検証システムの構成を示す。

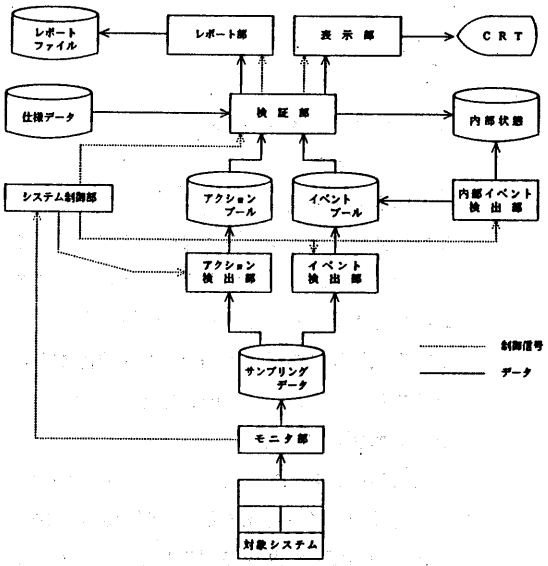

図9　システム構成

6　検証手法の評価および考察

本手法を評価するために、開発試作段階の製品に、いくつかの誤りを埋め込み、それらが検出されるかどうか実験した。このとき、検出された誤りのうち、代表的なものを以下に示す。

・システムの異常動作

・内部タイマーの異常

これらの誤りは、従来の方法では検出するのに多くの時間を要した。特に、内部タイマーについて

は検出するのが非常に困難なものであった。

また、矛盾と判定されたものの多くは、仕様記述の不足によるものであり、本手法を仕様の完全性を調べる一つの手段と考えることもできる。

ところで、このような検証手法では与えられた入力系列に対してのみ等価性をいうことができるが、将来において矛盾となったかもしれないという問題がある。しかしながら、この手法は時間に関して非常に敏感であり、もし矛盾が存在するならば局所的に矛盾でなくても大局的には矛盾となる。このため、記述された仕様の時間的なスケールに対して、十分長い入力系列を要求することによって、この問題を解決することができる。

7 効果

本手法の適用に伴う効果としては、次のようなものがあげられる。

（1）検証の自動化

試験対象システムを任意に操作するだけで、そのときのシステムの振舞いが正しいかどうかを機械的に判定することができるようになった。これにより製品試験における人的な誤りを排除することができるとともに、工数の省力化にもつながる。

（2）信頼性の定量化

有限状態機械モデルに基づいて、いくつかのテスト十分性の基準を定義することができる。例えば、状態網羅度、状態遷移網羅度などである。このようなテスト十分性基準を導入することにより、システムの機能的な信頼性を定量化することが可能になった。

（3）設計の可視化

従来、自然言語により記述されていたシステムの設計仕様を有限状態機械モデルに基づく形式言語により記述することにより、図的な表現が可能になった。

（4）デバッグ支援

本手法では、矛盾と判断された状況に対する理由付けが可能になり、多くのデバッグ情報を提供することができる。

8 対象システム

本手法は、この例に限らず以下の特徴を持つシステムに適用可能である。

（1）イベント駆動

本手法は、外界や内界の刺激に対して、継続的に対応するような制御を行うもの対象としている。つまり、イベントの発生を認識して、そのときのシステムの状態に応じて、動作を決定するようなシステムについては適用することができる。

（2）内部イベント

また、本手法はシステムの内部状態を観察することが困難であるものについても適用することができる。例えば、システム時間の経過に基づく制御を行うようなシステムでも適用可能となっている。

ただし、本手法ではシステムの仕様は有限状態機械モデルにより表現できるものでなければならない。

9 おわりに

本稿では、システム設計に対して仕様を動的に検証する手法を提案した。この手法では、検証を機械的に行うことが可能であり、システムテストを高度に自動化することが可能である。

また、本手法により正しさの証明を行なうためには、システムの全入力空間においてその等価性を確認しなければならないが、これは現実には不可能である。そこで、テスト十分性の基準として定義した網羅度を統計的に評価して、検証精度を向上させることが今後の課題である。

参考文献

[1] West,C.H.:"General technique for communication protocol validation",IBM J.Res. Dev.,22,1(1978)

[2] 内平他: ソフトウェア設計におけるプロトタイピング, ｂｉｔ臨増, pp.166-179, (1984)