2008－SE－161 (6)
2008／9／26

記号モデル検査を用いた状態マシン図とシーケンス図の無矛盾性の検証

宮崎 仁，瀬古 剛一，横川 智教，佐藤 洋一郎，早瀬 道芳

概 要

本論文では，UML の状態マシン図とシーケンス図で記述されたシステムを対象とし，システムの動作とモジュール間のモデルから求めるメッセージ通信順序の無矛盾性を検証する手法を提案している．提案手法では記号モデル検査ツールSMV を用いて検証を行う．そのため，まず，状態マシン図とシーケンス図からシステム全体の動作モデルを表す論理式を生成し，次に，シーケンス図からモジュール間のメッセージ通信の順序関係を表す論理式を生成する手法を提案している．さらに，検証する性質である設計間の無矛盾性のCTL 表現を与えている．最後に，例題システムに提案手法を適用し検証することで，本手法の有効性，妥当性を確認している．

Symbolic Model Checking of Consistency between Statemachine diagrams and Sequence diagrams

Hisashi MIYAZAKI, Koichi SEKO, Tomoyuki YOKOGAWA, Yoichiro SATO, and Michiyoshi HAYASE

Abstract

In this paper, we propose a method to verify consistency between the transition relation and the sequence of software systems described by UML : statemachine diagrams and sequence diagrams. In order to use the symbolic model checker SMV, we first represent the transition relation of the system as a boolean formula, and then represent the sequence of messages exchanged between modules as a boolean formula. We also provided CTL formulas representing the consistency between the transition relation and the sequence. Finally, we show the availability of the proposed method by applying it to the watchdog mechanism.

1 まえがき

ソフトウェアシステムの開発ではシステムを様々な側面からモデル化するために，形式的手法であるUML (Unified Modeling Language) [1] が用いられている．UML は13種類の図を用いて設計を行うが，この場合，単一の UML 図で記述したモジュールの設計には誤りが無くても，複数種類の UML 図で記述した設計を組み合わせて動作させた場合にのみ誤りを生じる場合がある．この種の誤りを検出するには，設計に用いられる複数の図を統合して，システム全体の動作を検証しなければならない．

本論文では，状態マシン図およびシーケンス図を用いて記述されたソフトウェアシステムの設計の無矛盾性を，記号モデル検査ツールSMV(Symbolic Model Verifier) [2] を用いて検証するための手法を提案する．まず，状態マシン図から各モジュールの動作モデルを求めてそれらを統合し，論理式として表現する．次に，シーケンス図からモジュール間のメッセージ通信順序のモデルを求め，同様に論理式として表現する．動作モデルとメッセージ通信順序モデルとの無矛盾性を時相論理 CTL によって表現することで，SMV による検証を実現する．

2 UML

ここではUML の13種類の図の中から，本論文で検証の対象としている状態マシン図とシーケンス図

について述べる．

2.1 状態マシン図

状態マシン図 $SM = (S, T, S_{init}, S_{fin}, name)$ は以下によって定義される．

- $S = \{s_1, s_2, \cdots\}$：状態の集合
- $T = \{t_1, t_2, \cdots\}$：遷移の集合
- $S_{init} \subseteq S$：初期状態の集合
- $S_{fin} \subseteq S$：終了状態の集合
- $name$：オブジェクトの名前

状態 s_i は以下の 1 つの要素を持つ．

- $name$：状態の名前

遷移 t_i は以下の 5 つの要素を持つ．

- evt：イベント
- grd：ガード条件
- act：アクション
- $src \subseteq S$：遷移の前状態
- $tgt \subseteq S$：遷移の後状態

イベントは遷移のきっかけであり，ガード条件は遷移を有効にする論理式であり，そしてアクションは遷移時に行われるメッセージ送信などの処理である．

例えば，図 1 の状態マシン図は以下のように定義される．

$S = \{s_1, s_2\}$，$T = \{t_1, t_2\}$，$S_{init} = \{s_1\}$，$S_{fin} = \{\phi\}$，$name =$ "$life1$"，
$name(s_1) =$ "$state1$"，$name(s_2) =$ "$state2$"，
$evt(t_1) =$ "syn"，$grd(t_1) = \phi$，$act(t_1) = \phi$，
$src(t_1) = s_1$，$tgt(t_1) = s_2$，
$evt(t_2) = \phi$，$grd(t_2) = \phi$，$act(t_2) =$ "ack"，
$src(t_2) = s_2$，$tgt(t_2) = s_1$．

2.2 シーケンス図

シーケンス図 $SQ = (L, M, O)$ は以下によって定義される．

- $L = \{l_1, l_2, \cdots\}$：ライフラインの集合
- $M = \{m_1, m_2, \cdots\}$：メッセージの集合

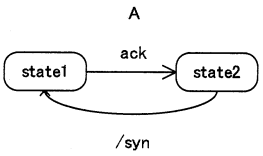

図 1: 状態マシン図

- $O = \{o_1, o_2, \cdots\}$：オカレンスの集合

ライフラインはメッセージの送受信に関わるオブジェクトであり，オカレンスはメッセージの送信または受信といった一つ一つの処理である．

ライフライン l_i は以下の 2 つの要素を持つ．

- $name$：ライフラインの名前
- $occ \subset O$：ライフライン上のオカレンスの集合

メッセージ m_i は以下の 4 つの要素を持つ．

- $name$：メッセージの名前
- $snd \in O$：送信オカレンス
- $rcv \in O$：受信オカレンス
- $sort = \{s, a\}$：メッセージの種類（s：同期，a：非同期）

メッセージは同期メッセージと非同期メッセージに分けられる．同期メッセージでは，メッセージの送信側はメッセージの受信処理が完了してから次の処理を行う．非同期メッセージでは，メッセージの送信側はメッセージの送信処理の直後に次の処理を開始する

オカレンス o_i は以下の 4 つの要素を持つ．

- $msg \in M$：オカレンスで処理されるメッセージ
- ll：オカレンスが存在するライフライン
- num：オカレンスのライフライン上での順序
- $type = \{s, r\}$：オカレンスの種類（s：送信，r：受信）

例えば，図 2 のシーケンス図は以下のように定義される．

$L = \{l_1, l_2\}$，$M = \{m_1, m_2\}$，$O = \{o_1, o_2, o_3, o_4\}$，
$name(l_1) =$ "A"，$occ(l_1) = \{o_1, o_2\}$，$name(l_2) =$ "B"，$occ(l_2) = \{o_3, o_4\}$，

$name(m_1) = "syn"$, $snd(m_1) = o_1$, $rcv(m_1) = o_3$, $sort(m_1) = s$,
$name(m_2) = "ack"$, $snd(m_2) = o_2$, $rcv(m_2) = o_4$, $sort(m_2) = a$,
$msg(o_1) = m_1$, $ll(o_1) = l_1$, $num(o_1) = 1$, $type(o_1) = s$,
$msg(o_2) = m_2$, $ll(o_2) = l_1$, $num(o_2) = 2$, $type(o_2) = r$,
$msg(o_3) = m_1$, $ll(o_3) = l_2$, $num(o_3) = 1$, $type(o_3) = r$,
$msg(o_4) = m_2$, $ll(o_4) = l_2$, $num(o_4) = 2$, $type(o_4) = s$.

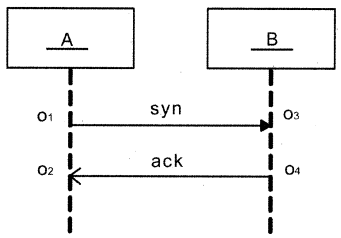

図 2: シーケンス図

3 UMLの論理式表現

SMV による検証を行うため，まず，与えられた図を元にシステムの構成要素を変数として定義する．次に，状態マシン図の記述を元に，状態遷移モデルを表す論理式 $\mathcal{B}$ を求める．さらに，シーケンス図の記述を元に，メッセージ順序モデルを表す論理式 $\mathcal{S}$ を求める．これらの論理式を元に，SMV の入力となる論理式記述を求める．最後に，検証すべき性質を CTL 上の時相論理式を用いて表現する．

3.1 システムの構成要素

ここでは，オブジェクトの状態と，それらがやり取りするメッセージをシステムの構成要素として考える．

状態マシン図のどの状態がアクティブであるかは，その状態マシンの全ての状態名を定義域とするような多値変数 $state = \{name(s_1), name(s_2), \cdots\}$ として表現する．この変数は，状態マシン図のアクティブである状態の名前を値としてもつ．

例えば，図 1 の状態マシン図のそれぞれのオブジェクトが持ちうる変数は以下のようになる．

$A = \{"state1", "state2"\}$.
$B = \{"state1", "state2"\}$.

全てのオブジェクトの初期状態がアクティブならば，それぞれのオブジェクトの状態を表す変数は

$A = "state1"$,
$B = "state1"$,

となる．

メッセージは，そのメッセージが生起しているときに真となる論理変数として表現する．しかし，状態マシン図内のイベントやアクションにおけるメッセージ名はシステム全体においてユニークであるとは限らない．そのため，全てのメッセージ msg にそのメッセージを受信するオブジェクト名を連接することで，システム内でユニークな論理変数 $obj.msg$ として表現する．

例えば，図 1 の遷移がもつメッセージはイベント syn とアクション ack である．ここで syn はオブジェクト A によって受信されるので，このメッセージは論理変数 $A.syn$ と表現する．ack はオブジェクト B によって受信されるので，このメッセージは論理変数 $B.ack$ と表現する．このようにしてユニークな変数とすることができる．

以上の手順で求められた変数の集合が V となる．

3.2 状態遷移モデルを表す論理式 $\mathcal{B}$

システム全体の遷移関係を表す論理式を求めるため，まず，状態マシン図の全ての遷移 $t_i \in T$ を論理式によって表現する．遷移 t_i は，イベント $evt(t_i)$，ガード条件 $grd(t_i)$，アクション $act(t_i)$，前状態 $src(t_i)$，そして後状態 $tgt(t_i)$ によって定義される．これらの

それぞれの要素を論理式として表現することで，遷移 t_i の論理式による表現を得る．

前状態 $src(t_i)$ は，以下のように論理式として表現する．この状態マシンの状態を表す変数を $state$ とすると，$src(t_i)$ がアクティブであることは，論理式 $\mathcal{B}_{src} \equiv (state = name(src(t_i)))$ として表現する．

後状態 $tgt(t_i)$ についても同様に，論理式 $\mathcal{B}_{tgt} \equiv (state' = name(tgt(t_i)))$ として表現する．

イベント $evt(t_i)$ は，その生起は $evt(t_i)$，破棄は $\neg evt(t_i)'$ でそれぞれ表現できるので，論理式 $\mathcal{B}_{evt} \equiv evt(t_i) \wedge \neg evt(t_i)'$ として表現する．

ガード条件 $grd(t_i)$ は，そのままガード条件を表す論理式として用い，論理式 $\mathcal{B}_{grd} \equiv grd(t_i)$ として表現する．

アクション $act(t_i) = \{msg_1, msg_2, \cdots\}$ は，メッセージ $msg_1, msg_2, \cdots$ を生起させる処理であるので，論理式 $msg_1' \wedge msg_2' \wedge \cdots$ で表現できる．従って，アクションは，論理式 $\mathcal{B}_{act} \equiv (msg_1' \wedge msg_2' \wedge \cdots)$ として表現する．

また，遷移 t_i に関連しない変数 $x_1, x_2, \cdots \in V$ については，現状態と次状態で値が変化しないことを明示する必要がある．これは，論理式 $\mathcal{B}_{inv} \equiv (x_1' = x_1) \wedge (x_2' = x_2) \wedge \cdots$ として表現する．

遷移 t_i は上記の論理式の論理積 $\mathcal{B}_i \equiv \mathcal{B}_{src} \wedge \mathcal{B}_{tgt} \wedge \mathcal{B}_{evt} \wedge \mathcal{B}_{grd} \wedge \mathcal{B}_{act} \wedge \mathcal{B}_{inv}$ として表現する．

例えば，図 1 の $state1$ から $state2$ への遷移の各要素は，以下のように論理式として表現する．

$\mathcal{B}_{src} \equiv (A = \text{"}state1\text{"})$,
$\mathcal{B}_{tgt} \equiv (A' = \text{"}state2\text{"})$,
$\mathcal{B}_{evt} \equiv A.ack \wedge \neg A.ack'$,
$\mathcal{B}_{inv} \equiv (B' = B) \wedge (syn' = syn)$.

全ての状態マシン図の全ての遷移 t_i に対してこのように $\mathcal{B}_i$ を求めた後，これらを全て論理和で結合した論理式を $\mathcal{B} \equiv \bigvee \mathcal{B}_i$ とする．論理式 $\mathcal{B}_i$ は，遷移 t_i が実行されたときかつそのときのみ真となるため，論理式 $\mathcal{B}$ が真となるのは，状態マシン図に含まれるいずれかの遷移が行われたときかつそのときのみである．従って，このようにして求めた論理式 $\mathcal{B}$ によって，状態マシン図で記述されたシステム全体の遷移関係を表す．

3.3 メッセージ順序モデルを表す論理式 $\mathcal{S}$

メッセージ通信の順序関係を表す論理式 $\mathcal{S}$ は，シーケンス図によって記述されたメッセージ通信の順序に従った遷移が行われたときに真となる論理式である．

メッセージ通信の順序関係を論理式として表すため，シーケンス図のライフライン $l_i \in L$ 上の全てのオカレンス $o_j \in occ(l_i)$ を，シーケンス図の順序関係に従ってオカレンスが実行されるときのみ真となる論理式として表現する．順序関係を表現するためには，現在の時点でメッセージ通信がどの部分まで完了しているかを表す必要がある．そのため，シーケンス図の各ライフライン l_i について，l_i のオカレンスがどこまで完了したかを示す変数 $point_i$ を定義する．$point_i$ は値 $p_{i\cdot 0}, p_{i\cdot 1}, \cdots$ をもち，l_i の j 番目のオカレンスまでが完了していることを $point_i = p_{i\cdot j}$ で表す．また，l_i の全てのオカレンスが完了したことを $point_i = p_{i\cdot END}$ によって表す．$P = \{point_0, point_1, \cdots\}$ とする．

ライフライン l_i 上の j 番目のオカレンス o_j を，(1) 非同期メッセージの処理，(2) 同期メッセージの送信の処理，(3) 同期メッセージの受信の処理，の 3 つの場合について，それぞれ以下の手順に従い論理式 $\mathcal{S}_{i\cdot j}$ として表現する．

(1) o_j が非同期メッセージの処理の場合

o_j が実行されるということは，(a) 現在の状態で $j-1$ 番目までのオカレンスが完了しており，かつ (b) o_j に対応するメッセージ通信が行われ，かつ (c) 次の状態で j 番目までのオカレンスが完了することである．(a) および (c) は，それぞれ定義から論理式 $\mathcal{S}_{pre} \equiv (point_i = p_{i\cdot j-1})$ および論理式 $\mathcal{S}_{post} \equiv (point_i' = p_{i\cdot j})$ で表現できる．(b) は以下のようになる．まず，メッセージ名が $name(msg(o_j))$ であり，メッセージを受信するライフライン名が $name(rcv(msg(o_j)))$ であるから，このメッセージは論理変数 $name(rcv(msg(o_j))).name(msg(o_j))$ で表される．o_j が送信ならばオカレンスによりメッセージが生起するため，(b) は，論理式

$\mathcal{S}_{occ} \equiv name(rcv(msg(o_j))).name(msg(o_j))$ で表現できる．o_j が受信ならばオカレンスによりメッセージが破棄されるため，(b) は，論理式 $\mathcal{S}_{occ} \equiv \neg name(rcv(msg(o_j))).name(msg(o_j))$ で表現できる．$point_m(m \neq i)$ の値が変化しないことは論理式 $\mathcal{S}_{inv} \equiv (point'_1 = point_1) \wedge (point'_2 = point_2) \cdots$ によって表現できるので，以上からライフライン l_i のオカレンス o_j の実行は，$\mathcal{S}_{i\cdot j} \equiv \mathcal{S}_{pre} \wedge \mathcal{S}_{occ} \wedge \mathcal{S}_{post} \wedge \mathcal{S}_{inv}$ と表現する．

例えば，図 2 のオカレンス o_4 は非同期メッセージ ack の送信であるので，

$\mathcal{S}_{pre} \equiv (point_2 = p_{2\cdot 1})$,
$\mathcal{S}_{post} \equiv (point'_2 = p_{2\cdot 2})$,
$\mathcal{S}_{occ} \equiv A.ack$,
$\mathcal{S}_{inv} \equiv (point'_1 = point_1)$,

となり，オカレンス o_2 は非同期メッセージ ack の受信であるので，

$\mathcal{S}_{pre} \equiv (point_1 = p_{1\cdot 1})$,
$\mathcal{S}_{post} \equiv (point'_1 = p_{1\cdot 2})$,
$\mathcal{S}_{occ} \equiv \neg A.ack$,
$\mathcal{S}_{inv} \equiv (point'_2 = point_2)$,

となる．

(2) o_j が同期メッセージの送信処理の場合

o_j が実行されるということは，(d) 現状態で $j-1$ 番目までのオカレンスが完了しており，かつ (e) o_j に対応する同期メッセージが送信され，かつ (f) 次状態では，j 番目のオカレンスは完了せず o_j に対応するメッセージが受信されるのを待つということである．(d) および (e) は，それぞれ上述した (a) の $\mathcal{S}_{pre}$ および (b) の $\mathcal{S}_{occ}$ と同様に表現できる．(f) は，ライフライン l_i が o_j に対応するメッセージの受信を待っていることを，$point_i$ の値 $\tilde{p}_{i\cdot j-1}$ によって表すことで，論理式 $\mathcal{S}_{post} \equiv (point'_i = \tilde{p}_{i\cdot j-1})$ によって表現できる．$point_m(m \neq i)$ の値が変化しないことは論理式 $\mathcal{S}_{inv} \equiv (point'_1 = point_1) \wedge (point'_2 = point_2) \cdots$ によって表現できるので，以上からライフライン l_i の o_j の実行は，$\mathcal{S}_{i\cdot j} \equiv \mathcal{S}_{pre} \wedge \mathcal{S}_{occ} \wedge \mathcal{S}_{post} \wedge \mathcal{S}_{inv}$ と表現する．

例えば，図 2 のオカレンス o_1 は同期メッセージ syn の送信であるので，

$\mathcal{S}_{pre} \equiv (point_1 = p_{1\cdot 0})$,
$\mathcal{S}_{post} \equiv (point'_1 = \tilde{p}_{1\cdot 0})$,
$\mathcal{S}_{occ} \equiv B.syn$,
$\mathcal{S}_{inv} \equiv (point'_2 = point_2)$,

となる．

(3) o_j が同期メッセージの受信処理の場合

o_j が実行されるということは，(g) 現状態で $j-1$ 番目までのオカレンスが完了しかつメッセージを送信したライフラインが受信を待っており，かつ (h) o_j に対応する同期メッセージが受信され，かつ (i) 次状態で j 番目までのオカレンスが完了しかつ送信側のオカレンスも完了するということである．送信側のライフラインを l_k，オカレンスを o_l とすると，(g) は，論理式 $\mathcal{S}_{pre} \equiv (point_i = p_{i\cdot j-1}) \wedge (point_k = \tilde{p}_{k\cdot l-1})$ によって表現できる．(h) は，上述した (b) の $\mathcal{S}_{occ}$ と同様に表現できる．(i) は，論理式 $\mathcal{S}_{post} \equiv (point'_i = p_{i\cdot j}) \wedge (point'_k = p_{k\cdot l})$ によって表現できる．$point_m(m \neq i,k)$ の値が変化しないことは論理式 $\mathcal{S}_{inv} \equiv (point'_1 = point_1) \wedge (point'_2 = point_2) \cdots$ によって表現できるので，以上からライフライン l_i の o_j の実行は，$\mathcal{S}_{i\cdot j} \equiv \mathcal{S}_{pre} \wedge \mathcal{S}_{occ} \wedge \mathcal{S}_{post} \wedge \mathcal{S}_{inv}$ と表現する．

例えば，図 2 のオカレンス o_3 は同期メッセージ syn の受信であるので，

$\mathcal{S}_{pre} \equiv (point_2 = p_{2\cdot 0}) \wedge (point_1 = \tilde{p}_{1\cdot 0})$,
$\mathcal{S}_{post} \equiv (point'_2 = p_{2\cdot 1}) \wedge (point'_1 = p_{1\cdot 1})$,
$\mathcal{S}_{occ} \equiv \neg B.syn$,

となる．

全てのライフライン l_i の全てのオカレンス o_j に対してこのように $\mathcal{S}_{i\cdot j}$ を求めた後，これらを全て論理和で結合した論理式を $\mathcal{S} \equiv \bigvee \mathcal{S}_{i\cdot j}$ とする．論理式 $\mathcal{S}_{i\cdot j}$ は，シーケンス図で与えられた順序関係に従ってオカレンス o_j が実行されたときのみ真となるため，論理式 $\mathcal{S}$ が真となるのは，シーケンス図で与えられた順序関係に従ってメッセージ通信が行われたときのみである．従って，この論理式 $\mathcal{S}$ によって，シーケンス図で記述されたメッセージの順序関係を表すことができる．

3.4 SMV の入力論理式の生成

3.2, 3.3 で生成した論理式 $\mathcal{B}$ と論理式 $\mathcal{S}$ を論理積として結合した論理式 $\mathcal{B} \wedge \mathcal{S}$ は，状態マシン図に記述された遷移がシーケンス図に記述されたメッセージの順序関係に従って行われるときのみ真となる．従って，この論理式を SMV の入力論理式として用いることで，SMV によりメッセージの順序関係を検証することが可能となる．

また，SMV で検証を行うためには，システムの初期状態を与えなければならない．初期状態は，集合 V と集合 P に含まれる変数の初期値として，以下のように定める．まず，状態を表す変数 $state$ の初期値は，状態マシン図によって定義された初期状態 S_{init} とする．状態マシン図によって初期状態が定められていない状態の初期値は不定とする．次に，メッセージの生起を表す変数の初期値は全て偽とする．最後に，変数 $point_i$ の初期値は $p_{i\cdot 0}$ とする．

3.5 CTL 式の生成

本論文では，設計間の無矛盾性，すなわち，状態マシン図で記述されたシステムがシーケンス図で与えられたメッセージ通信の順序関係を満たすことをシステムが満たすべき性質として考える．

ここでは，シーケンス図で与えられたオカレンスが全て実行されたときのみ，システムはメッセージ通信の順序関係を満たすと定義する．この特性を CTL 式を用いて記述すると以下のようになる．

$$
\begin{aligned}
& EF(point_1 = p_{1\cdot END}) \\
\wedge \quad & EF(point_2 = p_{2\cdot END}) \\
\wedge \quad & \cdots
\end{aligned}
$$

ここで $point_1, point_2, \cdots$ は 3.3 節で定義したライフラインのオカレンスの完了を表す変数である．$point_i = p_{i\cdot END}$ となるのはライフライン l_i のオカレンスが全て完了したときのみである．この CTL 式が真ならば，システムがシーケンス図で与えられたメッセージ通信の順序関係を満たすということが示される．

4 検証例

watchdog mechanism[3] に提案手法を適用し，Intel Core2 Duo 3.00GHz の CPU と 2GB RAM のメモリをもつ計算機上で検証を行った．

4.1 Watchdog mechanism

watchdog mechanism は，4 つの状態マシン図（図 3）と 1 つのシーケンス図（図 4）として与えられる．

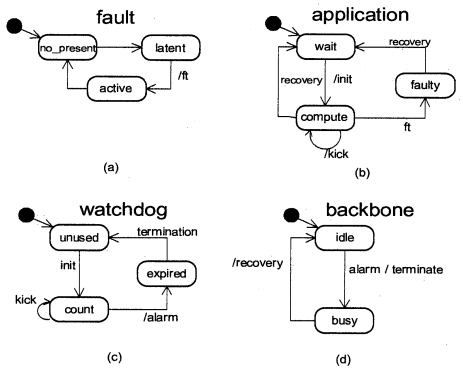

図 3: 状態マシン図 (Watchdog mechanism)

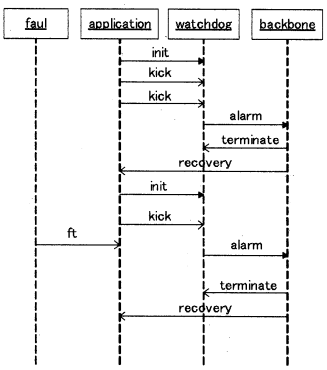

図 4: シーケンス図 (Watchdog mechanism)

提案手法に従ってこれらのUML図から論理式を生成し，SMVによる検証を行った結果，このシステムはシーケンス図で与えられたメッセージ通信の順序関係を満たすことが確認できた．

また，図3(b)の状態マシン図を図5のように変更し，同様に検証を行った結果，誤りを検出することができた．図4のシーケンス図には，*application*から*watchdog*へメッセージ*kick*が送信するメッセージ通信順序が記述されているが，図5の状態マシン図にはメッセージ*kick*を伴う遷移が存在しないので，シーケンス図のメッセージ通信順序は満たされない．このように状態マシン図とシーケンス図を組み合わせて検証することでシステムの遷移関係とメッセージ通信の順序関係の間に矛盾がある場合にはそれを検出できることを確認した．

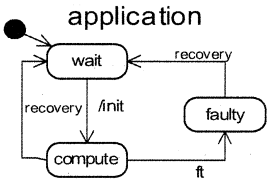

図5: 変更した状態マシン図 (Watchdog mechanism)

5 まとめ

本論文では，複数種のUML図で記述されたシステム設計を対象に記号モデル検査を用いて検証を行う手法を提案した．提案手法を用いることで，状態マシン図とシーケンス図で記述されたシステム設計の無矛盾性を検証することができる．

また，提案手法をwatchdog mechanismに適用し，状態マシン図で記述されたシステムの動作がシーケンス図で記述されたメッセージ通信順序を満たさない場合は，それを検出できることを確認した．

しかし，大規模システムを対象とした場合，到達可能状態数が増加し，SMVによる検証に要する時間は大きくなる．検証時間を削減するため，現在は，有界モデル検査[4]の適用について検討を進めている．

謝辞

本研究の一部は，科学研究費補助金若手研究(B)19700030の助成を受けている．

参考文献

[1] O. M. Group: "Unified Modeling Language", Object Management Group (2001). http://www.uml.org.

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

[3] S. Bernardi, S. Donatelli and J. Merseguer: "From uml sequence diagrams and statecharts to analysable petrinet models", Workshop on Software and Performance, pp. 35–45 (2002).

[4] A. Biere, A. Cimatti, E. M. Clarke and Y. Zhu: "Symbolic model checking without bdds", TACAS (Ed. by R. Cleaveland), Vol. 1579 of Lecture Notes in Computer Science, Springer, pp. 193–207 (1999).