プログラミング 18－10
(1998. 3. 23)

通信プロトコルの回復処理における設計誤りの Real-Time Temporal Logic を用いた検出

長野 伸一　角田 良明　菊野 亨

大阪大学大学院基礎工学研究科情報数理系専攻

〒 560-8531 大阪府豊中市待兼山町 1-3

Phone: 06-850-6568　Fax: 06-850-6569
E-mail: {s-nagano, kakuda, kikuno}@ics.es.osaka-u.ac.jp

あらまし フォールトトレランス性とリアルタイム性の2つを備えた通信プロトコルをリスポンシブプロトコルと呼ぶ．本稿では，Real-Time Temporal Logic を用いた通信プロトコルのリスポンシブネス検証と適用実験について述べる．通信プロトコルがリスポンシブネス性を満たすために必要な条件を Real-Time Temporal Logic を用いて論理式で記述できるので，回復処理における設計誤りの検出を効率良く行うことができる．次に，検証実験ではクライアント・サーバモデル上の通信システムにおけるコネクション確立のためのプロトコルを実際に設計し，Real-Time Temporal Logic を用いて設計誤りを検出できることを示す．

キーワード 通信プロトコル，リスポンシブネス，検証，Real-Time Temporal Logic，モデルチェッキング

Detection of Design Faults Using Real-Time Temporal Logic in Responsiveness Verification of Communication Protocols

Shin'ichi Nagano, Yoshiaki Kakuda and Tohru Kikuno

Department of Informatics and Mathematical Science
Graduate School of Engineering Science, Osaka University

1-3 Machikaneyama-cho, Toyonaka-shi, Osaka 560-8531, Japan

Phone: 06-850-6568　Fax: 06-850-6569
E-mail: {s-nagano, kakuda, kikuno}@ics.es.osaka-u.ac.jp

Abstract In this paper, we propose a protocol verification method using Real-Time Temporal Logic for responsive communication protocols. This verification method checks fault-tolerant property and real-time property of a given protocol. Since conditions required for responsiveness can be described by formulas of Real-Time Temporal Logic, design faults in the protocol can be detected effectively. Next, we design a protocol for connection establishment on a communication system based on the client-server model. Then by applying the proposed method to the connection establishment protocol, we can detect design faults of the protocol and show the usefulness of the proposed method.

Key words communication protocol, responsiveness, verification, Real-Time Temporal Logic, model checking

1 まえがき

近年フォールトトレランス性とリアルタイム性を同時に満たす通信プロトコルの開発への期待が高まってきている [13, 14]. 例えばプラント制御システムはサーバサイトとクライアントサイトからなる構成で実現され, 相互にメッセージを迅速に送受信するために両サイト間でコネクションを常に張っておく必要がある [4, 9]. そのためどちらかのサイトに故障が発生してシステムが異常状態に陥った場合, 再び迅速にコネクションを確立して正常状態に回復することが強く要求される. このようにフォールトトレランス性とリアルタイム性の2つの性質を満たす通信プロトコルをリスポンシブプロトコルと呼ぶ [7].

リスポンシブプロトコルを設計する場合, フォールトの発生とそれに伴う異常状態からの回復を常に考慮しなければならない. しかしフォールトがどのサイトにどのタイミングで発生するかに関して数多くの状況を網羅し, 各状況に対する回復処理を決めていく作業では, 状況の想定洩れや処理の記述ミスが混入する可能性が非常に高い [6, 8].

リスポンシブプロトコルの検証では, フォールトの発生によって任意の異常状態に陥っても許容時間以内に必ず正常状態へ回復できることを証明することが本質である. ところが一般に異常状態の数は正常状態数よりも非常に多いので, プロトコル設計者が全ての異常状態の系列を1つ1つ調べて通信プロトコルの設計誤りを見つけ出すのでは非常に時間がかかってしまう. また, 従来の設計誤りを検出する手法 [1, 3, 5] は, 検出の対象が正常処理のみである, そもそも実時間を扱うことができない, あるいは実時間を扱うことができても幾つかの厳しい制限が加えられている, などの理由で実用的ではない.

本研究では, 回復処理における設計誤りを効率良く検出することを目的として Real-Time Temporal Logic を用いたモデルチェッキング手法 [1, 2] を利用する. まず, 通信プロトコルがリスポンシブネス性を満たすために必要な条件を Real-Time Temporal Logic を用いて論理式ϕ で与える. 論理式ϕには, 時間の経過に伴うシステム状態の変化を実時間を考慮して記述することが可能である. 次に拡張有限状態機械で記述された通信プロトコルに対して, 文献 [11] の手法を適用して実時間に基づいた異状態系列を生成する. 最後に各異常状態系列に対して論理式ϕの充足可能性の判定を行う. その結果として, リスポンシブネス性を満たしていない原因となる設計誤りを絞り込むことができる.

また, Real-Time Temporal Logic を用いたモデルチェッキング手法を設計誤り支援ツールとして作成する. 次にコネクション確立プロトコルを例に取り上げ, 検証ツールをコネクション確立プロトコルへ適用した事例を示す. そのために, まず回復処理を含めてコネクション確立プロトコルを実際に設計した. ツールを正攻法で設計したコネクション確立プロトコルへ適用した結果, あるタイミングでフォールトが発生した場合に適切なメッセージが送信されず正常状態へ回復できないという設計誤りが検出された. また, 検証ツールの実行時間は非常に小さく, コネクション確立プロトコルのような実用規模の通信プロトコルに十分適用できることが分かった.

本稿の構成は次の通りである. まず2節で本研究を通じて例として取り上げるコネクション確立プロトコルについて説明する. 次に, 3節でリスポンシブネス検証の概要を述べる. 4節で Real-Time Temporal Logic の形式的定義を与えた後, 5節で設計誤り検出手法を述べる. 6節で検出手法をコネクション確立プロトコルへ適用した事例について報告する. 最後に7節でまとめと今後の課題について述べる.

2 通信プロトコルの設計

2.1 階層的記述

本稿では, 通信プロトコルをサイトレベル, プロセスレベル, 拡張有限状態機械レベルの3つに分けて考える. まずサイトレベルの記述では, 各サイトの構成をブラックボックスと見なしてサイト間の通信のみに着目する. 次にプロセスレベルでは各サイトをプロセスとチャネルで構成する. 各プロセスはブラックボックスと見なしてプロセス間の通信のみに着目する. 最後に拡張有限状態機械レベルでは, 各プロセスとチャネルをそれぞれ 拡張有限状態機械, FIFO キューでモデル化する.

2.2 コネクション確立プロトコル

最近のプラント制御システムは, 一般にクライアント・サーバモデルに基づいて実現されている. クライアント・サーバモデル上のネットワークシステムでは, クライアントサイトとサーバサイトの各組がメッセージを送受信する準備が整っていること, すなわち両サイト間でコネクションが確立されていることをお互いが認識する必要がある. コネクションを確立するために両サイト間で行われる一連のメッセージのやりとりのことを, 通常コネクション確立プロトコルと呼ぶ.

以降では, コネクション確立プロトコルのサイトレベルでの動作について述べる.(プロセスレベル, 拡張有限状態機械レベルの動作は文献 [12] を参照).

図1は6個のサイトと10個のチャネルから構成された通信システムを表しており, サイト間の各チャネルにおけるメッセージ転送には10単位時間かかることを表している. 本稿では図1を通信システムの例として取り上げ, クライアントサイト $Site_0$ とサーバサイト $Site_i$ $(1 \leq i \leq 5)$ 間でコネクションを確立することを考える.

まず, コネクション確立の状況を表す次の3つの値を定義する.

START: 現在コネクションは確立されておらず, かつ確立要求も発生していない状態.

WAIT: 確立要求が発生した後で, かつ実際にコネクションが確立される前の状態.

ESTABLISHED: コネクションが確立された状態.

次に, $Site_0$ と $Site_i$ $(1 \leq i \leq 5)$ 間のコネクション確立プロトコルのサイトレベルの記述を示す (図2は $Site_0$ と $Site_i$ 間のコネクション確立を示している). ここで,

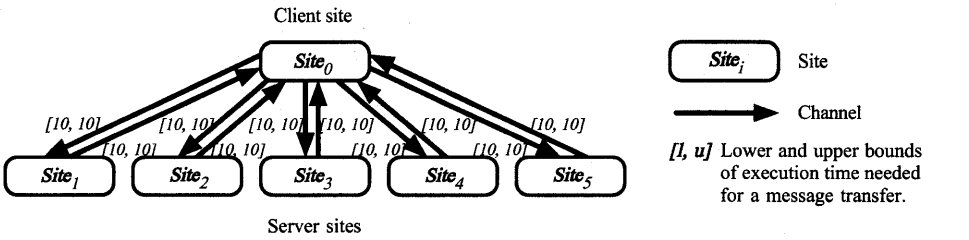

図 1: ネットワークの例

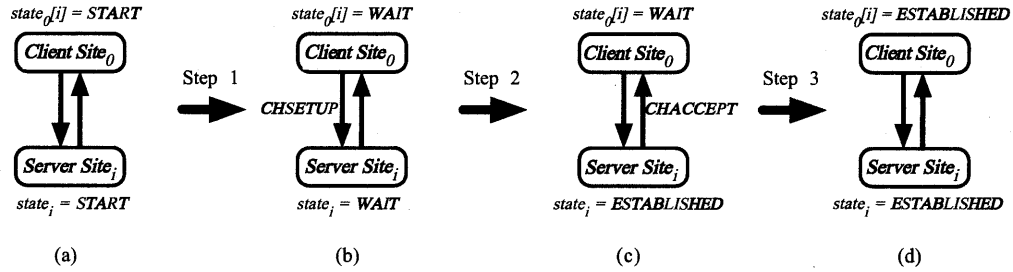

図 2: コネクションの確立

変数 $state_0[i]$ と $state_i$ は サイト $Site_0$, $Site_i$間のコネクションの状態を保持する変数で，$START, WAIT, ESTABLISHED$ のいずれかの値をとる．$state_0[i]$ はクライアントサイト $Site_0$ のローカル変数，$state_i$ はサーバサイト $Site_i$ のローカル変数である．初期値は $state_0[i] = state_i = START$ である (図 2(a))．

Step 1: まず，クライアントサイト $Site_0$ でサーバサイト $Site_i$ に対するコネクション確立要求が発生すると，$Site_i$ へコネクション確立要求メッセージ $CHSETUP$ を送信し，$state_0[i] = WAIT$ とする．次に，サーバサイト $Site_i$ では 確立要求メッセージを受信する準備が整うと，$state_i = WAIT$ とする．

Step 2: サーバサイト $Site_i$ は $CHSETUP$ を受信すると，サイトの負荷やネットワークの状態を調べる．コネクションの確立が可能であれば，$Site_0$ にメッセージ $CHACCEPT$ を送信し，$state_i = ESTABLISHED$ とする．コネクションの確立が困難であれば，$CHREJECT$ を送信し，$state_i = START$ とする．

Step 3: クライアントサイト $Site_0$ が $CHACCEPT$ を受信した場合に限ってコネクションは確立され $state_0[i] = ESTABLISHED$ とする．一方，$CHREJECT$ を受信した場合，コネクション確立要求を棄却し $state_0[i] = START$ とする．

$Site_0$ と $Site_i$ 間でコネクション確立要求が発生する前は $(state_0[i], state_i) = (START, START)$，確立を目指して両サイト間でメッセージをやりとりしている最中は $(state_0[i], state_i) = (WAIT, WAIT)$，確立されると $(state_0[i], state_i) = (ESTABLISHED, ESTABLISHED)$ となる (図 2を参照)．

3 リスポンシブネス検証

本節ではリスポンシブプロトコルの形式的な定義を与えた後，文献 [11] で提案したリスポンシブネス検証法の概要について述べる．

3.1 リスポンシブネス条件

定義 1 (システム状態) 通信プロトコル $\mathcal{P}$ の時刻 T におけるシステム状態 ss は $ss = (gs, ge)$ と定義される．ここで gs はグローバル状態であり，各プロセスが到達している状態と変数の値から構成される．一方，$ge = \{\langle e_1, l_1, u_1\rangle, \ldots, \langle e_q, l_q, u_q\rangle\}$ である．各 l_p, u_p $(1 \leq p \leq q)$ は正の整数で，イベント e_p の実行が完了するために必要な時間 (残存時間と呼ぶ) の下限と上限を表す．

例 1 システム状態の例を図 3に示す．2 行目の $state_0[1] = WAIT$ より $Site_0$ が $Site_1$ とコネクションを確立中であることが分かる．実際，プロセス $P_{m,0}$ から $P_{r,1}$ へのチャネルでは確立要求メッセージ $CHSETUP$ が転送中である (17行目の $(m_{m0,r1}, id_{m0,r1}) : m_{m0,r1} = CHSETUP \wedge id_{m0,r1} = 1$)．

また，$state_0[i] = START$ $(2 \leq i \leq 5)$ (2,3 行目) より他のサーバサイト $Site_i$ に対しては確立要求メッセージは送信されていない．一方，サーバサイト $Site_j$ $(1 \leq j \leq 5)$ ではクライアントサイトからのコネクション

確立要求を待っている状態にいる (7,9,11,13,15 行目の $state_j = WAIT$).

更に，ss_{27} では 5 つのイベントが実行中である (19,20,21 行目).

システム状態 ss_{i+1} が ss_i の次状態であるとき，$ss_i \vdash ss_{i+1}$ と表す [10].

定義 2 (状態系列) 通信プロトコル $\mathcal{P}$ のシステム状態の系列 $ss_1 ss_2 ss_3 \ldots ss_n$ が条件 $ss_i \vdash ss_{i+1}$ $(1 \leq i \leq n-1)$ を満たすとき，$\mathcal{P}$ の状態系列 $\sigma = ss_1 ss_2 ss_3 \ldots ss_n$ と定義する.

初期状態 ss_0 から到達可能なシステム状態を正常状態と呼ぶ．正常状態でないシステム状態を異常状態と呼ぶ．正常状態の集合と異常状態の集合をそれぞれ $\mathcal{S}_{normal}$, $\mathcal{S}_{abnormal}$ と表す.

定義 3 (リスポンシブプロトコル) 通信プロトコル $\mathcal{P}$ が次のリスポンシブネス条件を満たすとき，$\mathcal{P}$ はリスポンシブプロトコルであると定義する．ここで，R は非負整数であり，回復許容時間と呼ぶ.

リスポンシブネス条件 : $\forall ss_i \in \mathcal{S}_{abnormal}$

(1) ss_i から正常状態に到達できる.

(2) ss_i からの最大回復時間は R 以下である.

3.2 リスポンシブネス検証法

リスポンシブネス検証への入力は次の 4 つである：(1) 拡張有限状態機械でモデル化された通信プロトコル，(2) 各イベント e_i の実行時間の上限値 u_i と下限値 l_i，(3) 初期システム状態 ss_0，(4) 回復許容時間 R. 一方，出力は次の 3 つである：(1)$\mathcal{S}_{normal}$，(2)$\mathcal{S}_{abnormal}$，(3) プロトコル $\mathcal{P}$が R単位時間以内に回復できるか否かの判定結果.

リスポンシブネス検証法の概要を次に示す (詳細は文献 [11] を参照).

Step 1: 正常状態の集合 $\mathcal{S}_{normal}$ を求める.

Step 2: 未定義遷移によって生じる異常状態を求め，その集合を $\mathcal{S}^0_{abnormal}$ とする．また，$\mathcal{S}_{abnormal} = \mathcal{S}^0_{abnormal}$ とする.

Step 3: 各 $ss_i \in \mathcal{S}^0_{abnormal}$ から到達可能なシステム状態を生成し，集合 $\mathcal{S}_{abnormal}$ に入れる.

Step 4: $\mathcal{S}_{abnormal}$ と R に対してリスポンシブネス条件を満たすか否かをチェックする.

4 Real-Time Temporal Logic

命題論理の真偽値が時間の経過とともにどのように変わるかを定性的に記述し検証するための形式的体系の 1 つとして Temporal Logic(時相論理) がある．しかし，Temporal Logic の記述能力は時間の定性的な変化にとどまっており，プラント制御システムに要求されるようなハードリアルタイム性に関する論理式を記述することができない．そこで，時間の変化を定量的に記述できるように Temporal Logic を拡張した Real-Time Temporal Logic が提案されてきており，活発に研究が進められている [2].

Real-Time Temporal Logic の体系は次に示すような種々の観点から分類できる：(1) 命題論理か 1 階述語論理か，(2) 線形か分岐的か，(3) 離散時間か連続時間か，(4) 過去時制か未来時制か．本稿で扱う Real-Time Temporal Logic は離散時間を用いた命題線形論理であり，時間演算子は未来時制に関するもののみを含んでいる.

次に，Real-Time Temporal Logic の構文規則と意味解釈を定義する [2]

まず，直観的には Real-Time Temporal Logic は通常の命題論理に次の 3 つの時間演算子を追加したものである：$\Box_{[l,u]}$(always), $\Diamond_{[l,u]}$(eventually), $\mathcal{U}_{[l,u]}$(until) (lと u は $l \leq u$ を満たす非負整数)．例えば，p を原子命題と仮定すると，論理式 $\Box_{[0,4]}p$ は "現在の時刻から 4 単位時間後まで常に p が成り立つ" ことを意味する.

定義 4 (構文規則) Real-Time Temporal Logic の論理式の集合は，次の 3 つの規則で生成されるものだけを含む.

(1) 各原子命題 p は論理式である.

(2) ϕ が論理式ならば，$\neg\phi$ と (ϕ) は論理式である.

(3) ϕ_1 と ϕ_2 が論理式ならば，$\phi_1 \wedge \phi_2$ と $\phi_1 \mathcal{U}_{[l,u]} \phi_2$ は論理式である.

関係演算子 $\phi_1 \vee \phi_2$ と $\phi_1 \rightarrow \phi 2$ は，それぞれ $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\neg\phi_1 \vee \phi_2$ と定義される．また，時間演算子 $\Diamond_{[l,u]}\phi$, $\Box_{[l,u]}\phi$ は，それぞれ，$true\ \mathcal{U}_{[l,u]}\phi$, $\neg\Diamond_{[l,u]}\neg\phi$ と定義される.

例 2 論理式の例として次のϕ_1を考える.

$$\begin{aligned}\phi_1 \quad = \quad & \Box_{[0,\infty]}(state_0[1] = WAIT \rightarrow \\ & \Diamond_{[0,90]}(state_0[1] = ESTABLISHED \\ & \wedge\ state_1 = ESTABLISHED))\end{aligned}$$

ϕ_1 は時刻 0 以降任意の時刻で次のことが成り立つことを意味している: "論理式 $state_0[1] = WAIT$ が成り立つ時刻から数えて 0 単位時間以上 90 単位時間以内に論理式 $state_0[1] = ESTABLISHED \wedge state_1 = ESTABLISHED$ が成り立つ."

定義 5 (論理式の意味解釈) 論理式 ϕ が状態系列 σ に対して時刻 T で真となることを $(\sigma, T) \models \phi$ と表し，次のように再帰的に定義する.

(1) p が原子命題のとき，$(\sigma, T) \models p$ iff 時刻 T における σ 上のシステム状態 ss_i で p が真となる

(2) $(\sigma, T) \models \neg\phi$ iff $(\sigma, T) \not\models \phi$

(3) $(\sigma, T) \models (\phi_1 \wedge \phi_2)$ iff $(\sigma, T) \models \phi_1$ and $(\sigma, T) \models \phi_2$

(4) $(\sigma, T) \models \phi_1 \mathcal{U}_{[l,u]} \phi_2$ iff $\exists T'(T + l \leq T' \leq T + u)[(\sigma, T') \models \phi_2]$ and $\forall T''(T \leq T'' < T')[(\sigma, T'') \models \phi_1]$

$$
\begin{aligned}
&ss_{27} = \\
&(\ (\langle received : m_{m,0} = CHOPEN \wedge id_{m,0} = 2 \wedge state_0[1] = WAIT \wedge state_0[2] = START \wedge \\
&state_0[3] = START \wedge state_0[4] = START \wedge state_0[5] = START \wedge tout_0[1] = 0 \wedge \\
&tout_0[2] = 0 \wedge tout_0[3] = 0 \wedge tout_0[4] = 0 \wedge tout_0[5] = 0\rangle, \\
&\langle start1 : m_{a,0} = CHSUCCEED \wedge id_{a,0} = 5\rangle, \langle wait3 : m_{q,0} = CHACCEPT \wedge id_{q,0} = 1\rangle, \\
&\langle received : m_{t,0} = CHSTART \wedge id_{t,0} = 1\rangle, \langle wait3 : m_{r,0} = CHACCEPT \wedge id_{r,0} = 1\rangle, \\
&\langle idle1 : m_{m,1} = CHSETUP \wedge id_{m,1} = 1 \wedge state_1 = WAIT\rangle, \langle wait2 : m_{a,1} = CHSUCCEED \wedge id_{a,1} = 1\rangle, \\
&\langle wait : m_{q,1} = CHSETUP \wedge id_{q,1} = 1\rangle, \langle wait : m_{r,1} = CHSETUP \wedge id_{r,1} = 1\rangle, \\
&\langle idle1 : m_{m,2} = CHSETUP \wedge id_{m,2} = 2 \wedge state_2 = WAIT\rangle, \langle wait2 : m_{a,2} = CHSUCCEED \wedge id_{a,2} = 2\rangle, \\
&\langle wait : m_{q,2} = CHSETUP \wedge id_{q,2} = 2\rangle, \langle wait : m_{r,2} = CHSETUP \wedge id_{r,2} = 2\rangle, \\
&\langle idle1 : m_{m,3} = CHSETUP \wedge id_{m,3} = 3 \wedge state_3 = WAIT\rangle, \langle wait2 : m_{a,3} = CHSUCCEED \wedge id_{a,3} = 3\rangle, \\
&\langle wait : m_{q,3} = CHSETUP \wedge id_{q,3} = 3\rangle, \langle wait : m_{r,3} = CHSETUP \wedge id_{r,3} = 3\rangle, \\
&\langle idle1 : m_{m,4} = CHSETUP \wedge id_{m,4} = 4 \wedge state_4 = WAIT\rangle, \langle wait2 : m_{a,4} = CHSUCCEED \wedge id_{a,4} = 4\rangle, \\
&\langle wait : m_{q,4} = CHSETUP \wedge id_{q,4} = 4\rangle, \langle wait : m_{r,4} = CHSETUP \wedge id_{r,4} = 4\rangle, \\
&\langle idle1 : m_{m,5} = CHSETUP \wedge id_{m,5} = 5 \wedge state_5 = WAIT\rangle, \langle wait2 : m_{a,5} = CHSUCCEED \wedge id_{a,5} = 5\rangle, \\
&\langle wait : m_{q,5} = CHSETUP \wedge id_{q,5} = 5\rangle, \langle wait : m_{r,5} = CHSETUP \wedge id_{r,5} = 5\rangle, \\
&\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \langle (m_{m0,r1}, id_{m0,r1}) : m_{m0,r1} = CHSETUP \wedge id_{m0,r1} = 1\rangle, \\
&\epsilon, \epsilon), \\
&\{\langle (m_{m0,r1}, id_{m0,r1}), 7, 7\rangle \ in\ C_{m0,r1}, \langle send(CHSETUP, id_{m,0})\ to\ P_{r,2}, 1, 1\rangle \ in\ P_{m,0}, \\
&\langle send(CHOPEN, id_{q,0})\ to\ P_{q,0}, 1, 1\rangle \ in\ P_{a,0}, \langle receive(m_{a,0}, id_{a,0})\ from\ P_{a,0}, 0, 0\rangle \ in\ P_{q,0}, \\
&\langle starting\_timer(tv_0[1], TIMEOUT), 1, 1\rangle \ in\ P_{t,0}\}\ )
\end{aligned}
$$

図 3: システム状態の例

$(\sigma, 0) \models \phi$ が成り立つ時かつその時に限り σ はϕ を充足可能であると言う.

5 モデルチェッキングによる設計誤り検出

3.2 節のリスポンシブネス検証法では通信プロトコル $\mathcal{P}$がリスポンシブネス条件を満たすか否かをチェックするに留まっている. リスポンシブネス条件の判定結果が偽となった場合には, $\mathcal{P}$ の仕様に設計誤りが混入している可能性がある. しかし, 異常状態の数は正常状態の数よりも非常に多いため異常状態集合 $\mathcal{S}_{abnormal}$ だけを用いて設計誤りを検出することは一般に難しい.

異常状態系列からの設計誤り検出を支援するために, 本稿では Real-Time Temporal Logic を用いる. 通信プロトコル $\mathcal{P}$ がリスポンシブネス性を満たすために必要な条件を Real-Time Temporal Logic を用いて論理式ϕ で与える. 各異常状態系列と ϕ に対してモデルチェッキングを行う. その判定結果をもとに設計誤りを絞りこむことが可能である.

入力は次の 2 つである : (1)3.2 節の手法で生成された異常状態の集合 $\mathcal{S}_{abnormal}$, (2)Real-Time Temporal Logic で記述された論理式 ϕ. 一方, 出力は ϕ の充足可能性の判定結果 ($true$ or $false$) である.

検証手順

Step 1: $\mathcal{S}_{abnormal}$ から異常状態系列の集合 $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ を求める.

Step 2: 各 σ_i $(1 \leq i \leq n)$ に対して $(\sigma_i, 0) \models \phi$ を判定する.

6 検証実験

6.1 検証ツールの開発

5 節に述べた検証手順のうち Step 2 の充足可能性判定を行う検証ツールを Perl で開発した (約 840 行).

入力は (1) 異常状態系列の集合 $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$, (2)Real-Time Temporal Logic で記述された論理式 ϕ, の 2 つである. 一方, 出力は (1) 各σ_i $(1 \leq i \leq n)$ に対する ϕ の充足可能性の判定結果 ($true$ or $false$), (2)ϕ が偽となる原因となったシステム状態, の 2 つである.

6.2 実験の概要

本実験の目的は, コネクション確立プロトコルの実際の設計工程において検証ツールを適用し, 設計誤りの検出, 特定ができることを確認することである.

設計の最終的な目標は 3.1 節のリスポンシブネス条件を満たすコネクション確立プロトコルを作成することである. リスポンシブネス条件を満たすためにはコネクション確立プロトコルの任意の異常状態系列が 4 節の論理式 ϕ_1 を充足する必要がある. ϕ_1 を充足しない系列に対しては次の 2 点に着目して論理式を作成し, 設計誤りの検出を試みる: (1) 適切なメッセージの送受信が行われているか否か, (2) 回復処理に要する時間が設計時の見積り値より小さいか否か.

また, 本実験を行うにあたって次の 2 つの仮定をおく.

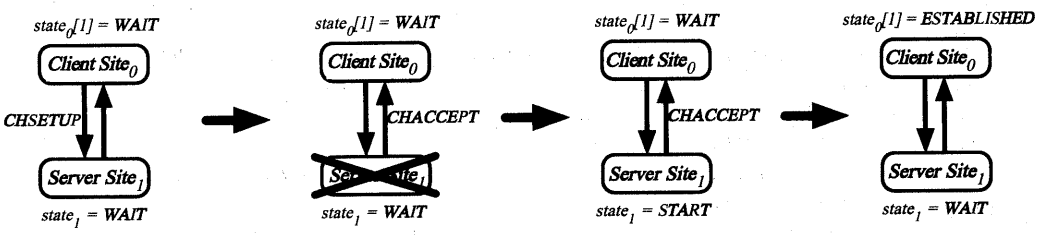

図 4: 正常状態に回復できない例

仮定 1 (サイトの起動に関する仮定) 図 1の全てのサイト $Site_i$ $(0 \leq i \leq 5)$ は時刻 0 で同時に起動する.

仮定 2 (フォールトに関する仮定) フォールトの種類はサイトの一時的故障とメッセージ消失のみを扱う.

実際に行った検証作業を, 設計フェーズ, シミュレーションフェーズ, 設計誤り検出フェーズに分けて説明する.

6.3 設計

まず, フォールトが発生しない場合に各プロセスで実行する処理を 2.1 節で述べた手順に忠実に記述した. 一方, フォールトが発生した場合の回復処理は, フォールトが発生した状況を想定してそれに応じた処理を記述した.

6.4 シミュレーション

3.2 節の手法を実装したシミュレータ [12] を使ってシミュレーションを行った. その結果, 正常状態の系列を順にたどって全てのコネクションの確立を確認できた.

次に, 各異常状態に対するシミュレーション結果から, 次の状況のフォールトが発生した場合の異常状態系列 σ_{30} ではコネクションが確立された状態へ到達できなかった. その状況とは, サーバサイト $Site_1$ からクライアントサイト $Site_0$ へコネクション確立の受理を意味するメッセージ $CHACCEPT$ が送信された後に $Site_1$ でフォールトが発生することである.

6.5 設計誤りの検出

確認 1 まず, 異常状態系列 σ_{30} と 論理式 ϕ_1 に対して検証ツールを適用した. その結果として, σ_{30} は ϕ を充足できないこと, ϕ 中の部分論理式 $state_0[1] = ESTABLISHED \wedge state_1 = ESTABLISHED$ を満たすシステム状態が σ_{30} の中に存在しないことが分かった.

確認 2 $Site_1$ でのフォールト発生は $Site_1$ が $CHACCEPT$ を送信した後であることから, $Site_0$ は $CHACCEPT$ を受信してコネクションが確立されたと誤って判断してしまっている可能性がある. まず, このことを確認する. コネクション確立要求を受理するメッセージ $CHACCEPT$ が $Site_0$ に届くまでの間は $state_0[1] = WAIT$ である. $Site_0$ が $CHACCEPT$ を受信すると内部処理に 10 単位時間を要した後に $state_0[1] = ESTABLISHED$ とする. この動作を論理式で記述したのが次の ϕ_2 である.

$$\phi_2 = \Box_{[0,\infty]}(state_0[1] = WAIT \rightarrow \Diamond_{[0,10]}(m_{m,0} = ACCEPT \wedge \Box_{[0,\infty]}(state_0[1] = ESTABLISHED)))$$

σ_{30} と ϕ_2 にツールを適用した結果, $true$ を返した. したがって, 一旦 $state_0[1] = ESTABLISHED$ となると値が変わらないことが確認された.

確認 3 確認 2 の結果から, ϕ_1 を充足できない原因は ϕ_1 の中の論理式 $state_1 = ESTABLISHED$ を充足できないことであると考えられる.

この推測が正しいことを確認するために, $Site_1$ が再起動後に $Site_0$ からのコネクション確立要求を待ち続けているか否かを調べる. 再起動直後は $state_1 = START$ である. 内部処理に 10 単位時間を要した後, $state_1 = WAIT$ となる. この動作を論理式で記述したのが次の ϕ_3 である.

$$\phi_3 = \Box_{[0,\infty]}(state_1 = START \rightarrow \Diamond_{[0,10]}(state_1 = WAIT \wedge \Box_{[0,\infty]}(state_1 = WAIT)))$$

σ_{30} と ϕ_3 に検証ツールの適用した結果, $true$ を返した. したがって, $Site_1$ は $Site_0$ からの確立要求メッセージ待ちの状態のままであることが確認された.

6.6 設計誤りの特定

6.5 節の結果からσ_{30} でコネクションが確立できない理由は次のように考えられる: $Site_1$ は再起動後, $Site_0$ からのコネクション確立要求を待ち続ける. このとき $state_i = WAIT$ である. 一方, $Site_0$ は $CHACCEPT$ を受信してコネクションは確立されたものと判断してしまう. このとき $state_0[i] = ESTABLISHED$ である. このように両サイトで矛盾した状態に陥ってしまうので, コネクションを確立することができない.

また, 実際に異常状態系列 σ_{30} を調べてみた. 図 4(b) は $Site_1$ でフォールトが発生した状態を表している. $Site_1$ は再起動後 (図 4(c)), 確立要求待ちの状態に移っている (図 4(d)). 一方, $Site_0$ は $CHACCEPT$

を受信してコネクションが確立されたと誤って判断してしまっている (図 4(d)). σ_{30} は図 4の矛盾状態で終わっており，コネクションは確立されていなかった．

更に，プロトコル仕様を調べた結果，クライアントサイト $Site_0$ にはサーバサイトの故障発生を検知する処理が組み込まれていないこと，サーバサイト $Site_i$ ($1 \leq i \leq 5$) には再起動したことを $Site_0$ に通知する処理が組み込まれていないことを確認した．

6.7 検証ツールの評価

6.4節のシミュレーションは SUN UltraSPARC UA1 上で行った．生成された異常状態系列は 193 個で，各状態系列中のシステム状態の数は最も多いもので 174 個あった．また，6.5節でこれら全ての異常状態系列と論理式 ϕ_1 に対して検証ツールを適用した際，検証ツールの実行時間は 647 秒であった．この値は状態数が非常に多いにも関わらず検証ツールが高速に処理できたことを意味する．したがって，コネクション確立プロトコルのような実用規模の通信プロトコルに対しても検証ツールは十分適用できるものと予想される．

7 まとめ

本稿では，通信プロトコルのリスポンシブネス検証における回復処理の設計誤りの検出に Real-Time Temporal Logic を用いたモデルチェッキング手法を適用した．更に，モデルチェッキング手法を実用プロトコルへ適用した検証事例を示した．実用プロトコルの例としてコネクション確立プロトコルを取り上げ，回復処理も含めてコネクション確立プロトコルを設計した．検証の結果として，あるタイミングでフォールトが発生すると適切なメッセージが送受信されないために正常状態へ回復できなくなる設計誤りが検出された．

提案法に関する今後の研究課題としては，次の 2 つが考えられる: (1) 論理式の充足可能性判定に要する時間の削減技法の検討，(2) 検証に必要な論理式を自動生成する技術の開発．

参考文献

[1] R. Alur and D. Dill: "The theory of timed automata," *LNCS 600 Real-Time: Theory in Practice*, pp.45-73, 1992.

[2] R. Alur and T. A. Henzinger: "Logics and models of real-time: a survey," *LNCS 600 Real-Time: Theory in Practice*, pp.74-106, 1992.

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla: "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. on Programming Languages and Systems*, Vol.8, No.2, pp.244-263, 1986.

[4] D. Ferrari and D. C. Verma: "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, Vol.8, No. 3, pp.368-379, 1990.

[5] M. G. Gouda and N. J. Multari: "Stabilizing communication protocols," *IEEE Transactions on Computers*, Vol.40, No.4, pp.448-458, 1991.

[6] G. J. Holzmann: "Design and Verification of Computer Protocols," Prentice Hall, NJ, 1991.

[7] H. Kopetz and Y. Kakuda(eds.): "Responsive computer systems," Dependable Computing and Fault-Tolerant Systems, Vol.7, Springer-Verlag, pp.17-26, 1993.

[8] M. T. Liu: "Protocol Engineering," Advances in Computers, Vol.29, 1989.

[9] I. Mizunuma, C. Shen, and M. Takegaki: "Middleware of distributed industrial real-time systems on ATM networks," *Proc. of 17th IEEE Real-Time Systems Symposium*, pp.32-38, 1996.

[10] S. Nagano, Y. Kakuda, and T. Kikuno: "Timed reachability analysis method for communication protocols modeled by extended finite state machines," *Trans. of IPSJ*, Vol.37, No.5, pp.698-710, 1996.

[11] S. Nagano, Y. Kakuda, and T. Kikuno: "A new verification method using virtual system states for responsive communication protocols and its application to a broadcasting protocol," *IEICE Trans. on Fundamentals*, April 1998 (to appear).

[12] S. Nagano, Y. Kakuda, and T. Kikuno: "Experience of responsiveness verification for connection establishment protocols," *Proc. of 1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'98)* , April 1998 (to appear).

[13] 西尾 弘道, 安藤 隆, 福嶋 秀樹, 濱口 能任, 小本 孝則: "上下水道大規模監視制御システム," 三菱電機技報, Vol.69, No.12, pp.17-21, 1995.

[14] 竹垣 盛一: "実時間分散システム技術とその応用," 電子情報通信学会, 信学技報 IN96-81, 1996.