E-AoSAS++ における実行前検査のデバッグ支援環境に関する研究

2007MI025 古川 利彦　　2007MI176 野口 悠
指導教員　野呂 昌満

1　はじめに

本研究室では，E-AoSAS++[3] の研究をおこなっている．昨年度の研究 [4] では，E-AoSAS++ に基づいたシステムに対し，モデル検査ツール FDR[2] で検証する方法を提案した．モデル検査とは，状態遷移機械で表わされるシステムの振舞いを網羅的に走査して，システムが満たすべき振舞いを満たすか否かを自動的に検査する手法である．

E-AoSAS++ に基づいたシステムを FDR で検証する際，検証に失敗した際の FDR の出力から欠陥箇所を特定することは困難である．実行前検査の段階でのデバッグの過程が明確化されておらず，FDR から出力される反例と欠陥箇所の関係を明確にすることができないからである．

本研究の目的は，E-AoSAS++ のアーキテクチャ記述のデバッグ作業を省力化することである．アーキテクチャ記述のデバッグ作業とは，コードレベルで現れるであろう欠陥箇所を，コードを実行する前にアーキテクチャレベルで検査し，欠陥箇所を特定することである．アーキテクチャレベルで検査することができれば，コードレベルの欠陥を除去することができる．

本研究のアプローチは，デバッグの過程を定式化することである．デバッグ過程を定式化することで，デバッグ作業を自動化することができる．欠陥箇所の特定に有益な情報を提示するツールを作成することで，デバッグ作業の省力化をおこなう．

研究の進め方は次のとおりである．FDR から出力される反例を分析し，疑わしい状態を見つける．仕様と状態遷移機械の関係の分析をし，問題箇所を発見する．デバッグ支援に必要な情報，機能の考察をおこない，ツールの設計，作成をおこなう．実例を用いて，提案したデバッグ方法の妥当性を確認する．

本研究の結果として，デバッグ過程における疑わしい状態と問題箇所を定義した．デバッグに有益な情報を提示することで，デバッグ作業を省力化することができた．実例を用いることで，提案したデバッグ方法の妥当性を確認することができた．

2　背景技術

E-AoSAS++ の計算モデル，FDR を用いた検証について説明する．

2.1　E-AoSAS++ の計算モデルの概要

E-AoSAS++ の計算モデルの概要として，(1) 並行状態遷移機械，(2) イベントキューを介した非同期通信，について説明する．図 1 に E-AoSAS++ のモデルの概要を示す．

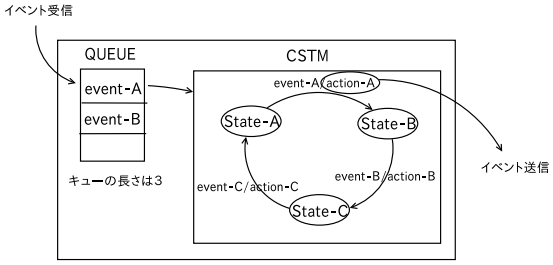

図 1　E-AoSAS++ の計算モデル

2.1.1　並行状態遷移機械

E-AoSAS++ に基づいたシステムは図 1 のように，並行に動作する状態遷移機械 (Concurrent State Transition Machine，以下 CSTM) の集合からなる．単一の CSTM の振舞いを説明する．

1. イベントを受信する
2. 受信したイベントに対応するアクションを実行する
   (a) イベントをほかの CSTM に対して通知する
   (b) イベントに応じて状態遷移をする

各 CSTM は非同期にイベントを送受信することで連携動作する．E-AoSAS++ では，イベントキューを用いることで非同期にイベントを送受信することを実現している．

2.1.2　イベントキューを介した非同期通信

E-AoSAS++ では図 1 に示すように，非同期通信を実現する方法として有限長のイベントキューを用いる．イベントを受信する際，キューが満杯でなければイベントを格納するが，キューが満杯の場合はイベントを格納することができない．また，イベントを取り出す際は，キューの先頭から順に受理可能なイベントを探索する．受理可能なイベントを見つけた場合，そのイベントをキューから取り出す．

2.2　FDR を用いた検証

本研究では，記述した CSP[1] を代表的なモデル検査ツール FDR で検査をおこなう．FDR は CSP で記述されたシステムの「仕様」と「実現：検証条件」間の詳細化関係を検査し，検査に失敗すると反例が出力される．

FDR が詳細化関係を用いて検証できる性質を表 1 に示す．安全性の検査では，「悪いことは決して起こらないこと」を確かる．活性の検査では，「良いことはそのうち起こること」を確かめる．

表 1　Refinement で検証できる性質

Refinement	検証できる性質
Trace refinement	安全性
Failure refinement	活性，デッドロックフリー性

3　デバッグの過程の分析

実際に E-AoSAS++ に基づいたシステムのデバッグをおこない，実行前検査の段階のデバッグ過程を明確化した．デバッグの過程を図 2 に示す．

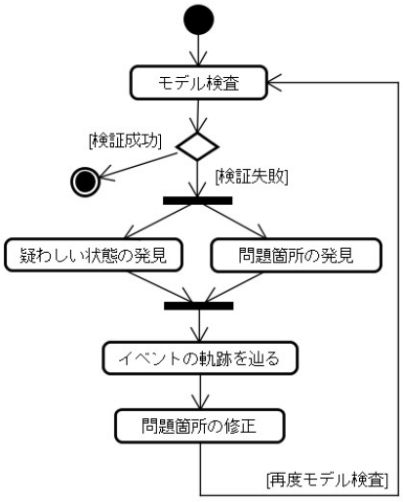

図 2　デバッグの過程

われわれが提案するデバッグ方法では，仕様が正しいという前提で，図 2 における疑わしい状態の発見と問題箇所の発見から，欠陥箇所を探す上で手がかりとなる出発点を決定する．その後，出発点からイベントの軌跡を辿り，欠陥箇所を特定する．図 2 の過程において重要と考える，(1) 疑わしい状態の発見，(2) 問題箇所の発見，についての説明をする．

3.1　疑わしい状態の発見

FDR の出力である反例から，疑わしい状態を見つける．疑わしい状態を見つけることで，どの CSTM に問題があるかの予測を立てる．実際に反例として検出された，疑わしい状態を示す．

E-AoSAS++ に基づいたシステムは，キューを介したイベントの送受信をおこなうので，疑わしい状態の候補として，キューの状態とイベントの軌跡，その他が挙げられる．

- キューの状態に着目
  - キューが満杯
  - キューにイベントが残っている
- イベントの軌跡に着目
  - イベントの軌跡が交差している
- その他
  - イベントの受信はするが，アクションが実行されていない etc...

キューの状態には，キューが満杯，キューにイベントが残ってる，キューが空の三つの状態が存在し，キューが満杯，キューにイベントが残っている場合，疑わしい状態である．イベントの軌跡には，特徴的な間違いとして，イベント実行順序の逆転があり，軌跡の交差として現れる．その他に，イベントの受信をしても，アクションが実行されていない場合に疑わしい状態だと判断できる．疑わしい状態に陥っていた場合，その箇所の CSTM が問題である可能性がある．ただし，必ずしも疑わしい状態である箇所が問題であるとは限らない．

3.2　問題箇所の発見

モデル検査をおこなう際に用いる，仕様と CSTM の関係から，問題箇所を発見する．仕様と CSTM の関係は安全性の検査と活性の検査の場合で異なる．関係の分析の仕方として，次のイベントに遷移することができるか否かで分類をおこなう．仕様と CSTM の関係について，(1) 安全性の検査の場合，(2) 活性の検査の場合，の二つの観点から示す．

3.2.1　安全性の検査の場合

安全性の検査の場合，仕様に遷移先が存在するか否かで，問題箇所となる CSTM の状態が変わる．図 3 では，仕様に遷移先が存在しない場合に，問題箇所として現れる CSTM の例を示す．

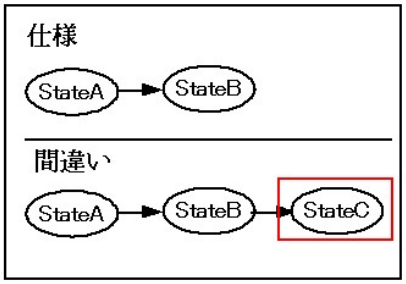

図 3　安全性の場合に問題となる状態 (遷移先が存在しない場合)

仕様に遷移先が存在しない場合に，問題がある状態として「遷移先が存在する CSTM」が挙げられる．図 3 に示したように，仕様では，StateB までしか遷移しないが，実現では，StateB 以降に遷移先が存在している場合が問題箇所である．

図 4 では，仕様に遷移先が存在する場合に，問題箇所として現れる CSTM の例を示す．

仕様に遷移先が存在する場合に，問題がある状態として「仕様の順番と異なる CSTM」や「遷移先が仕様と異なる CSTM」が挙げられる．図 4 に示したように，仕様では，StateA から始まり StateC まで遷移するが，実現では，StateB から始まる場合や，StateD のように遷移先が異なる場合が問題箇所である．

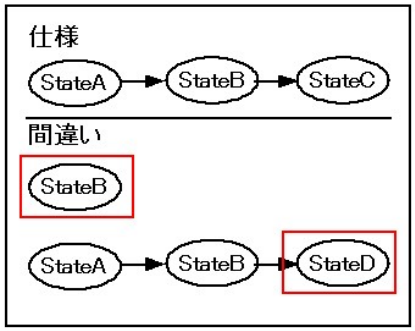

図 4　安全性の場合に問題となる状態 (遷移先が存在する場合)

3.2.2　活性の検査の場合の分析

活性の検査の場合の問題箇所として現れる CSTM の例を図 5 に示す．活性の検査の場合は，安全性の検査を満たしているという前提で検査をおこなう．これにより，安全性とは問題箇所となる CSTM の状態が異なる．

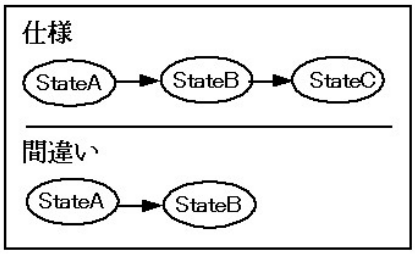

図 5　活性の場合に問題となる状態

活性の検査の場合に，問題がある状態として「遷移先が存在しない CSTM」が挙げられる．図 5 に示したように，仕様では，StateA，StateB，StateC と遷移するが，実現では，StateA から始まり StateB で停止している場合が問題箇所である．

以上の分析をもとに，欠陥箇所の特定を支援，省力化するツールの作成をおこなう．ツールについての説明は 4 章でおこない，5 章でデバッグ方法の妥当性の考察をおこなう．

4　デバッグ支援ツール

デバッグ作業を省力化するデバッグ支援ツールについて説明する．3 章で説明した疑わしい状態の候補，問題箇所の予測をするときに有益な機能を示す．

4.1　必要な機能

反例からデバッグをおこなう際に必要になる機能を示す．本研究では，以下の機能をデバッグ支援ツールで実行可能にする．ツールの機能を，(1) 疑わしい状態の発見，(2) 問題箇所の発見，(3) イベントの軌跡を辿る，の三つの点から示す．

4.1.1　疑わしい状態の発見に必要な機能

- イベントの軌跡の表示
- キューの状態の表示

4.1.2　問題箇所の発見に必要な機能

- 仕様の表示
- CSTM の状態の表示

欠陥箇所の特定に必要な機能として，3 章で説明した情報を提示する機能が必要である．これらの情報から，どの CSTM に問題があるかの予測がしやすくなる．

4.1.3　イベントの軌跡を辿る際に必要な機能

- 反例のシーケンス図の生成
- キューの状態の表示
- CSTM の状態の表示
- ステップ実行

欠陥箇所の特定を支援する際に，シミュレーション機能が有益である．シミュレーションでは，反例のシーケンス図とキューの状態，CSTM の状態を 1 ステップ毎に表示する．

反例のシーケンス図を生成することで，システム全体の振舞いが把握しやすくなる．ステップ実行によって，イベントの処理毎のキューの状態，CSTM の状態の変化を把握することができる．一連のプロセスの軌跡を表示することで，並行に動作している複数のイベントの軌跡から，一連の動作が把握しやすくなる．

4.2　デバッグ支援ツールの入力

デバッグ支援ツールの入力は，検証をおこなったシステムの仕様，環境，アーキテクチャのモデル検査をおこなった際に出力する反例である．有益な情報は，これらから得ることができる．

4.3　デバッグ支援ツールの出力

出力として，反例のシーケンス図の生成をおこなう．シーケンス図には，欠陥箇所特定の支援になるように有益な情報が付加されている．また，疑わしい箇所の指摘と，問題箇所の予測をツールが出力としておこない，出力の内容からデバッグをおこなう．

4.4　設計

シミュレーションと欠陥箇所の特定に必要な機能をもとに作成した，デバッグ支援ツールのクラス図を図 6 に示す．

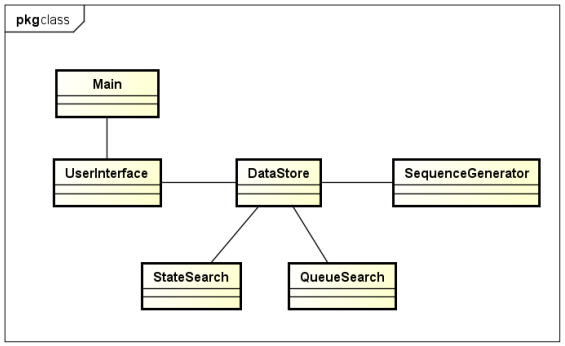

図 6　デバッグ支援ツールのクラス図

5　デバッグ方法の妥当性の考察

3 章で説明したデバッグ方法で，デバッグ作業の省力化の考察をおこなう．プリンタシステムの検証をおこなったときの反例を用いて，提案したデバッグ方法が妥当であるか考察する．

検証に失敗したシステムに対して，どれだけデバッグ方法を適用することができたかを示す．実際にデバッグをおこなった反例は 10 個である．適用できなかった反例については，本研究で考察した情報以外に必要な情報があると考える．詳しくは 5.3 節で説明する．

5.1　疑わしい状態の候補についての考察

実際に検証に失敗した反例 10 個全て，疑わしい状態を挙げた．疑わしい状態の候補である CSTM 自身が欠陥箇所であったものは 3 個である．疑わしい状態の候補からイベントを辿ることにより，欠陥箇所を特定することができたものは 5 個である．2 個は疑わしい状態とは別の所に原因があった．

5.2　問題箇所の予測についての考察

実際に検証に失敗した反例 10 個全て，問題箇所の予測から間違っている CSTM の予測をおこなった．そのうち，8 個の反例は，問題箇所を予測することができ，欠陥箇所は予測した所に存在した．2 個の反例は，判別できないパターンであった．

5.3　適用できなかった反例について

本研究で提案したデバッグ方法で，欠陥箇所特定の支援ができなかった反例についての考察をおこなう．デバッグ方法が適用できなかった原因として，同期の取り方に問題があった．適用できなかった反例を図 7 に示す．

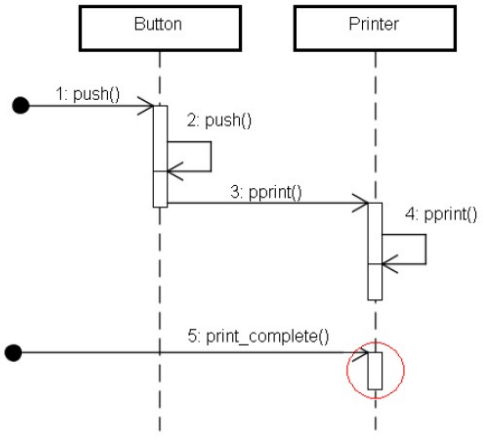

図 7　反例のシーケンス図

図 7 の場合，疑わしい状態の候補として，5.print_complete のイベントの受理ができないことが考えられる．しかし，問題箇所の予測をおこなう際に，仕様に対して問題のある CSTM が存在しなかったので，どの CSTM が怪しいかの予測をおこなうことができなかった．

上記のような関係の場合，問題箇所の予測をすることができないので，欠陥箇所の特定はできない．また，仕様と比べながらイベントの軌跡を辿っても問題のありそうな箇所を見つけることはできなかった．しかし，今回のシステムの場合，環境で同期を取っていることがわかった．同期の取り方に失敗しているのではないかと予測し，同期の取り方を修正したところ，検証に成功した．このことから，われわれが提案したデバッグ過程では情報が不足していることがわかった．また，ツールを用いて生成するシーケンス図だけでは，同期の情報を得ることができないことがわかった．

5.4　仕様が間違っている場合

検証をおこなうシステムの「仕様」か「実現：検証条件」のどちらかが正しいという前提条件がないと欠陥箇所の特定はできない．今回の研究では，仕様が正しいという前提でデバッグを支援する環境を確立したが，仕様が間違っている可能性も考えられる．仕様の間違っている箇所を特定するには，環境・アーキテクチャが正しいという前提で，関係を分析する必要がある．環境・アーキテクチャが正しいという前提がなければ，仕様の間違いである箇所を特定することはできない．

6　おわりに

本研究では，デバッグ支援ツールを作成することで欠陥箇所の特定の省力化を目指した．特定の省力化を目指すにあたって，デバッグの過程の明確化，反例の分析をおこない，欠陥箇所の特定に有益な情報を考察した．提案したデバッグ方法の妥当性について，実例を用いて考察した．

今後の課題として，今回欠陥箇所の特定ができなかった反例でもデバッグができるような環境を確立する必要がある．その際に，シーケンス図で出力する情報を整理し直す必要がある．また，仕様が間違っている場合を考慮した，デバッグ支援環境を確立する必要がある．

参考文献

[1] C.A.R. Hoare, *Communicating Sequential Process*, Prentice-Hall, 1985.

[2] Formal Systems(Europe), *“FDR2 User Manual,”* http://fsel.com/documentation/fdr2/html/fdr2manual.html, 2005.

[3] 加藤大地，蜂巣吉成，沢田篤史，野呂昌満，“アスペクト指向に基づくソフトウェアアーキテクチャの文書化方式，” 知能ソフトウェア工学研究会 *(KBSE)*, vol.108，no.449，pp.55-60，2009.

[4] 長谷川裕記，神谷浩翔，岡嶋智史，“E-AoSAS++ における実行前検査に関する研究，” 南山大学数理情報学部情報通信学科２００９年度卒業論文，2009.