Vol.2015-SLDM-170 No.13
Vol.2015-EMB-36 No.13
2015/3/6

同時性を考慮した並行システムの振舞い検証に関する考察

張 漢明[1,a)]　野呂 昌満[1,b)]　沢田 篤史[1,c)]

概要：並行システムのソフトウェア開発では，分散した複数のプロセスで同時に発生した事象を，漏れなく検知することが重要である．本研究の目的は，複数事象発生の同時性を考慮した，並行システム設計の振舞い検証を支援することである．本研究の基本的なアイデアは，並行システム全体の振舞いを区間の合成として定義し，同時性を区間内に局所化することである．本稿では，プロセス代数 CSP を用いて並行システムの同時性を定義して，同時性を考慮した振舞い検証の枠組みを提示する．同時性の仕様記述を一般化してモジュール化することにより，同時性に関する仕様と検証の記述を容易にする．

A discussion on concurrent system verification for considering simultaneous event occurences

Han-Myung CHANG[1,a)]　Masami NORO[1,b)]　Atsushi SAWADA[1,c)]

Abstract: In the software development of concurrent systems, it is important to detect the events without fail that occur simultaneously with distributed multiple processes. We aim to support the behavioral verification of concurrent system design considering the simultaneity of multiple events occurrence. Our basic idea is to define the behavior of an entire concurrent system as the synthesis of section, and to localize the simultaneity in the section. In this paper we define the simultaneity of concurrent systems using the process algebra CSP, and give the framework of the behavioral verification considering the simultaneity. Moralizing to generalize the specification methods for the simultaneity facilitate the descriptions of specifications and verification for concurrency.

1. はじめに

並行システムのソフトウェア開発では，システム全体の振舞いを網羅的に把握することは困難であり，開発の設計段階において振舞いを検証することが重要である．並行システムでは，並行性に起因する予期しない事象の発生順序により，原因究明が困難な障害となる場合がある [1]．このような予期しない事象による誤りを，プログラムの段階でテストによる検証で発見することは難しい．このような誤りを発見するための有効な手段として，モデル検査を用いた振舞い検証技術がある [2], [6]．モデル検査とはシステムの全ての有限状態を網羅的に調べて，仕様を自動的に検証する技術である．

[1] 南山大学理工学部ソフトウェア工学科
[a)] chang@nanzan-u.ac.jp
[b)] yoshie@nanzan-u.ac.jp
[c)] sawada@nanzan-u.ac.jp

並行システムでは，分散した複数のプロセスで同時に事象が発生することは避けられず，この同時に発生した複数の事象に対して正しく制御しなければならない．ソフトウェアによる制御では，このような並行プロセスにおける複数事象の発生を，逐次プロセスにおいて同期を用いて漏れなく記述する必要がある．同期を用いて正しく制御することは，並行システムのソフトウェア開発の困難さの 1 つである [3]．複数事象発生の制御の正しさを検証するためには，複数事象発生の同時性を厳密に定義することが最も重要である．

本研究の目的は，複数事象発生の同時性を考慮した，並行システム設計の振舞い検証を支援することである．複数事象の同時性に関連した概念を定式化してモジュール化することにより，検証のための振舞い仕様と検証式の再利用性の向上を目指す．抽象度の高いアーキテクチャ段階での設計では，通常，詳細な同時性は考慮せずに逐次の振舞い

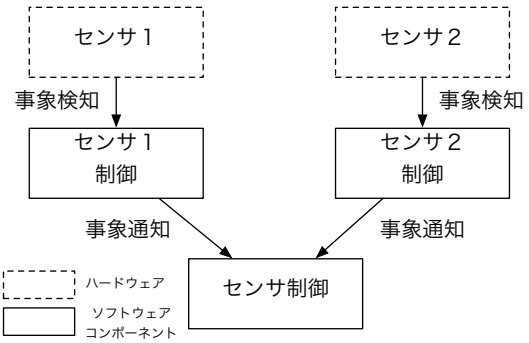

図 1 2 つのセンサーの制御
Fig. 1 Control of two sensors

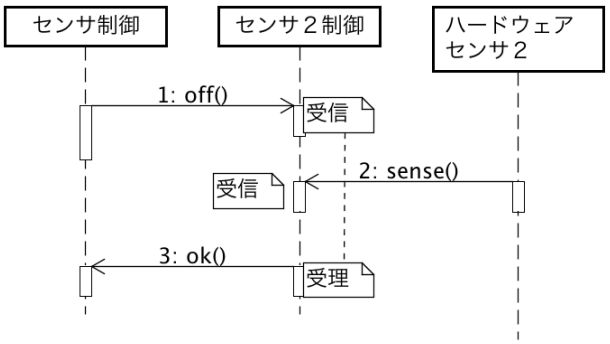

図 2 際どい事象発生順序の例
Fig. 2 An example of critical event occurrence order

を記述する．逐次の振舞いを検証するためには，同時に複数の事象が発生しないことを表現する必要があるので，逐次の振舞いは単純ではあるけれども同時性を考慮する必要がある．

本研究の基本的なアイデアは，並行システム全体の振舞いを区間の合成として定義し，同時性を区間内に局所化することである．区間の概念を用いて事象の同時性を記述するためのパターンを提示する．区間では，複数の事象が同時に発生することを表現するために，開始と終了を明示する．本研究では，同時に事象が発生することを，「ある区間内で複数の事象が発生すること」と定義する．システム全体の振舞いを，逐次，選択，繰返しの演算子を用いて逐次プロセスとして記述することにより，同時性の考慮については区間に集約される．

本稿では，プロセス代数 CSP[7], [9] を用いて並行システムの同時性を定義して，同時性を考慮した振舞い仕様記述法と振舞い検証の枠組みについて述べる．CSP は並行システムを形式的に記述するための言語であり，モデル検査器 FDR[4] を用いて振舞いを自動で検証することができる．自動販売機システムの事例を通して，提案する仕様記述法と検証法について再利用性の観点から議論する．

2. 基本的なアイデア

本研究で対象とする並行システムにおける複数事象発生の同時性の基本的なアイデアについて述べる．

2.1 並行システムにおける同時性

複数事象の同時発生の検知と，同時性検証の困難さについて説明する．

2.1.1 複数事象の同時発生の検知

2 つのセンサーをソフトウェアで制御する例を用いて，複数事象の同時発生の検知について説明する．図 1 は，センサ 1 とセンサ 2 のハードウェアをソフトウェアで制御する構造と事象通知を表している．1 つのセンサのハードウェアに，センサを制御するソフトウェアコンポーネント（センサ 1 制御，センサ 2 制御）があり，センサ制御ソフトウェアコンポーネントが 2 つのセンサを制御する．

2 つのセンサが同時に作動している時に，センサ 1 が事象を検知した場合，センサ 1 制御がセンサ制御に事象通知する．センサ制御はセンサ 1 制御からの事象通知を受けて，センサ 2 制御に事象検知がないかを調べる．先にセンサ 2 が検知した場合も同様にセンサ 1 の事象検知を調べる必要がある．また，3 つ以上の複数のセンサ制御の場合，これらの振舞いの発生事象のパターンを正確に把握する必要がある．

2.1.2 同時性検証の困難さ

上記のような同時性の検証の困難さについて説明する．

通知順序の逆転

センサを制御するコンポーネントが並行に動作していると，センサ 1 とセンサ 2 で発生した事象検知の順番が，センセ制御の事象通知と逆転する場合がある．センサ制御では，大域的な時間の概念がない場合，センサ 1 とセンサ 2 でどちらが先に発生したかを知ることができない．並行システムでは，事象検知の順番が正確に把握することができないときがあるので，同時の概念で事象を捉えることが適切な場合がある．

際どい事象発生順序

1 つのセンサからの事象通知を受けて，もう 1 つのセンセの作動を停止させる場合の，際どい事象の発生例について説明する．図 2 は，センサ制御からセンサ 2 制御にセンサ作動の停止（`off` 事象）を通知した際の，際どい事象発生の順序を表している．コンポーネント間の通信がキューを用いた非同期通信の場合，センサ 2 制御はキューに `off` 事象を受信し，その事象を受理したときにその結果をセンサ制御に通知する．

この受信と受理の間にセンサ 2 のハードウェアから `sense` 事象通知があった場合について考える．センサ 2 制御が `off` 事象を受理した時に，`sense` 事象がキューに受信された状態で在る．`off` 事象の受理の処理で，キューの状態を調べずにセンサ制御に通知すると，ハードウェアが事象を検知したことが漏れることになる．

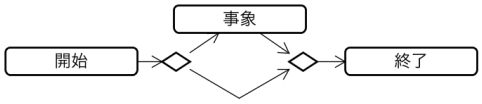

図 3 開始終了付き事象
Fig. 3 Event with start and end

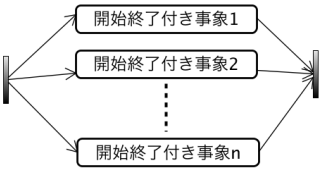

図 4 区間
Fig. 4 Section

この事象検知の漏れが例外処理などの重要な情報である場合，障害の原因となりうる．また，キューに残った事象がその後の振舞いに悪影響を与える可能性がある．このような際どい事象の発生は再現性が低く，プログラムの段階でテストにより発見することは困難である．

暗黙の前提条件

並行システムのソフトウェア開発では，通常，抽象度の高いアーキテクチャ段階では，複数事象の同時発生を考慮することはない．設計のある段階で同時性について設計されるが，同時の定義について厳密に定義されることはなく，同時の前提条件が暗黙の内に設定されている．

複数事象の同時性を検証するためには，同時の振舞いを厳密に記述しなければならない．検証者は同時を定義するための事象を分析する必要があり，これらの事象は入力系列の一部として表現される．同時性の定義が入力系列に埋もれると，保守の際に理解することが困難なので，複数事象の同時性をアーキテクチャ段階から仕様として明示することが，ソフトウェア開発の工程において重要である．

2.2 複数事象の同時性の概念

並行システムの全体の振舞いを区間の合成として表し，複数事象の同時性を区間として定義する．

2.2.1 区間

開始終了付き事象という概念を導入して，区間を開始終了付き事象を用いて定義する．

開始終了付き事象

開始終了付き事象の定義を以下に示す．

開始終了付き事象 =（開始，終了，事象）

開始終了付き事象は，開始，終了，事象の 3 項組として定義される．図 3 は開始終了付き事象の振舞いを UML のアクティビティ図で表したものである．事象は開始と終了の間で，発生するもしくは発生しないとして振舞う．

区間

区間の定義を以下に示す

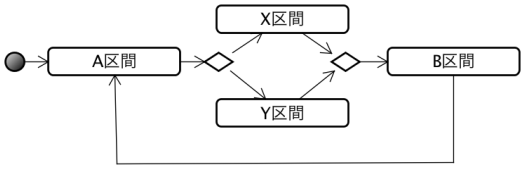

図 5 区間の合成の例
Fig. 5 An example of composition of section

区間 = 開始終了付き事象の集合

区間は開始終了付き事象の集合として定義される．図 4 は区間の振舞いを UML のアクティビティ図で表したものである．区間は開始終了付き事象のインターリーブとして振舞う．区間の開始は，開始終了付き事象のうちいずれかの開始で，終了は開始終了付き事象の終了が全て同期したときである．

2.2.2 区間事象発生

区間における振舞いとして，以下の 2 つの区間を定義する．

単一事象選択区間 区間で定義されている事象のうち，任意の事象が選択される．

複数事象発生区間 区間で定義されている事象のうち，指定した複数の事象が発生する．

2.2.3 区間の合成

並行システムの全体の振舞いを定義するために，区間の演算子を用意する．システムの振舞いは以下の演算子を用いて逐次プロセスとして表現する．

- 区間合成の演算子
  - 逐次，選択，再帰

区間を用いたシステムの振舞い例を図 5 に示す．まず，A 区間から始まり，X 区間もしくは Y 区間の後で，B 区間が実行される．システムはこの振舞いを繰り返す．それぞれの区間では，区間の開始と終了が定義されているので，区間にまたがって同時に事象が発生することはないことが保証される．

3. 同時性を含意した振舞い仕様記述法

プロセス代数 CSP を用いて，同時性を含意した振舞い仕様の記述法を定義する．CSP の記法は，モデル検査器 FDR で使われているテキスト形式の記法である CSP_M [7] を用いる．

3.1 区間

区間を開始終了付き事象を用いて定義する．

3.1.1 開始終了付き事象

開始終了付き事象の定義を以下に示す．

```
EVENT_WITH_START_AND_END(start,end,event) =
  start ->
    (event -> end -> SKIP [] end -> SKIP)
```

開始終了付き事象（EVENT_WITH_START_AND_END）は，開始（start），終了（end），事象（event）を引数とした関数として定義する．-> はイベントの逐次演算子，[] はプロセスの選択演算子である．SKIP はプロセスの正常終了を表している．

3.1.2 区間

区間の定義を以下に示す．

```
SECTION(s) =
  ||| (start, end, event):s @
    EVENT_WITH_START_AND_END
              (start, end, event)
```

区間（SECTION）は，開始終了付き事象を定義する 3 項組の集合 s を引数とした関数として定義する．||| はプロセス集合のインターリーブ演算子である．引数 s で指定した開始終了付き事象のプロセスがインタリーブで合成されることを表している．区間の終了は，開始終了付き事象の終了イベントが全て同期したときとなる．

3.2 区間事象発生

区間事象発生を定義する．

3.2.1 単一事象選択区間

単一事象選択区間の定義を以下に示す．

```
SEL_SECTION(s) =
  [] event:OccurredEvents(s) @ event -> SKIP
OccurredEvents(s) =
  {event | (start, end, event) <- s}
```

単一事象選択区間（SEL_SECTION）は，開始終了付き事象を定義する 3 項組の集合 s を引数とした関数として定義する．OccurredEvents は引数 s で指定した開始終了付き事象の事象の集合であり，SEL_SECTION は事象の任意選択のプロセスを表してる．

3.2.2 複数事象発生区間

複数事象発生区間の定義を以下に示す．

```
PL_SECTION(occurred, not_occurred) =
  (||| (st, end, ev):occurred @
            st -> ev -> end -> SKIP)
 |||
  (||| (st, end, ev):not_occurred @
            st -> end -> SKIP)
```

複数事象発生区間（PL_SECTION）は，事象が発生する開始終了付き事象の集合（occurred）と事象が発生しない集合（not_occurred）を引数とした関数として定義する．

3.3 区間の合成

システムの振舞いを定義するための区間の合成演算子，逐次，選択，再帰を以下に定義する．

逐次

```
SEQ(s1, s2) = s1; s2 -> SKIP
```

選択

```
SEL(ss) = [] s:ss @ s -> SKIP
```

再帰

```
P = P';P
```

逐次，選択は CSP の逐次演算子（->），選択演算子（[]）で表現され，再帰も CSP で用いられるプロセス名を用いて表される．

4. 同時性を考慮した振舞い検証の枠組み

同時性を考慮した振舞い検証の枠組みを提示する．

4.1 システムの振舞い

システムの振舞いを検証の対象と環境の並行合成で表現する．

```
システム = 検証の対象 [|SYN_ENV|] 環境
```

上記の右辺は，検証の対象と環境の並行合成を表している．SYN_ENV は並行合成する際の同期するイベントの集合である．環境では，検証の対象となる並行プロセスとのインタフェースを定義する．組込みシステムでは，環境はデバイスドライバの仕様の役割を果たす．

4.2 振舞い仕様

振舞い仕様は，プログラムのテストケースの概念と同様に，「入力」に対して期待される「出力」として定義する．入力を環境に局所化することにより，振舞い仕様の独立性を高める．

入力

入力はシステム全体の振舞いを区間の合成として定義して，入力をシステムの振舞いの制約として記述する．

```
CONSTRAINT(システム, SYN_SEC, 区間合成)
```

SYN_SEC は区間を定義する事象の集合である．

ここで制約とは，対象の振舞いを制約の振舞いで限定することを意味する．CSP では振舞いの制約を並行合成の演算子で表現することができる．

```
CONSTRAINT(p, sync, c) = p [|sync|] c
```

並行合成の左右のどちらも制約とみなすことができるが，ここではどちらが制約であるかを明示するために，第 3 引数の c を制約とみなす．

出力

入力時の出力の振舞いを定義する．出力は一般的には外部事象の入出力関係で定義される．

4.3 同時性を考慮した振舞い検証の検証式

CSP では仕様が実現を満たすことを，詳細化関係（Refinement）を用いて表現する [7]．CSP における仕様と実現の関係を以下に示す．

```
仕様 [FD= TARGET(実現，着目する事象)
```

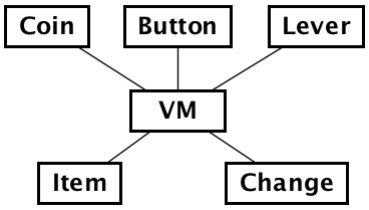

図 6 自動販売機の構造
Fig. 6 A structure of a vending machine

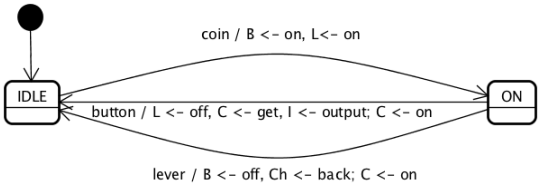

図 7 自動販売機の制御
Fig. 7 Vending machine controller

```
TARGET(S, targets) =
  S \ diff(Events, targets)
```

CSP では，仕様が実現を満たすことを，仕様が着目する事象に対して，仕様と実現が詳細化関係であると表現する．

同時性を考慮した振舞い検証の検証式の枠組みは

```
出力 [FD= TARGET(Constraint(システム,
                            SYN_SEC,
                            区間合成),
                 外部イベント)
```

となる．

5. 事例：自動販売機システム

図 6 に示す単純な自動販売機システムを用いて，区間を用いた検証例を示す．

5.1 区間合成を用いたシステムの振舞い

```
INPUT_PURCHASE =
  DEPOSIT; SELECT; INPUT_PURCHASE
```

入金区間では「コイン（COIN）」の事象があり，商品選択区間では「ボタン（BUTTON）」と「レバー（LEVER）」の事象がある．それぞれの区間の定義は以下のようになる．

```
DEPOSIT = SECTION({COIN})
SELECT = SECTION({BUTTON, LEVER})
```

COIN, BUTTON, LEVER はそれぞれ開始終了付き事象で 3 項組で定義されている．

5.2 逐次事象の振舞い検証

同時に事象が発生しないことを前提とした，逐次事象の振舞い検証例を示す．図 7 はそのような自動販売機の制御例である．

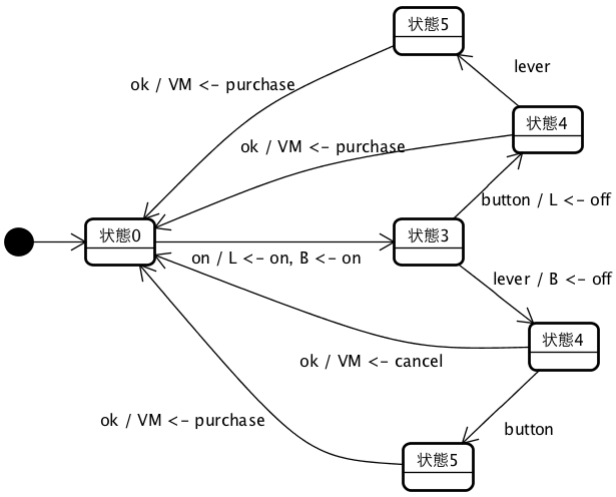

図 8 同時事象検知する状態遷移機械
Fig. 8 STM for detecting simultaneous events

入力

入力は各区間を単一選択区間で制約すれば良い．

```
SEQ_MODEL =
  CONSTRAINT(SYSTEM,
             SYN_SEC, SEQ_S)
SEQ_S =
  SEL_SECTION({COIN});
  SEL_SECTION({BUTTON, LEVER}); SEQ_S
```

出力

出力は，コイン挿入後に，ボタンが押されたら商品を排出し，レバーが引かれたらコインを返金する．

```
SEQ_SPEC = EXT_INSERTED;
  (EXT_PUSHED; EXT_ITEM []
     EXT_PULLED; EXT_CHANGE);
  SEQ_SPEC
```

検証式

検証式は以下のようになる．

```
SEQ_SPEC [FD=
    TARGET(SEQ_MODEL, ExternalEvents)
```

本節で示した検証式は，区間を用いた関数を用いて容易に記述され，その検証の意味も容易に理解できる．これらの関数を用いなければ，入力の記述は同期する事象が混在して，その理解が困難となる．

5.3 同時事象の振舞い検証

同時に事象が発生する同時事象の振舞い検証例を示す．図 8 は同時の事象発生に対応した制御例である．

入力

ここでは入力として，商品選択時にボタンとレバーの事象が同時に発生を繰り返す例を示す．

```
PL_MODEL =
  CONSTRAINT(SECTION_MODEL,
```

```
                ExternalControlEvents, PL)
  PL =
    PL_SECTION({COIN}, {});
    PL_SECTION({BUTTON, LEVER}, {}); PL
```

出力

図 8 では，ボタンとレバーが同時に発生した場合は，商品を排出する制御をしている．

```
  PL_SPEC = EXT_INSERTED;
    (EXT_PUSHED ||| EXT_PULLED); EXT_ITEM;
    PL_SPEC
```

検証式

検証式は以下のようになる．

```
  PL_SPEC [FD=
      TARGET(PL_MODEL, ExternalEvents)
```

6. 考察

同時性を含意した振舞い仕様記述法の有用性として，振舞い検証の再利用性と検証の指針について議論する．

6.1 振舞い検証の再利用性

前節で示した自動販売機の事例では，まず，図 7 に示した同時の事象を考慮していない販売制御のシステムに対して，単一事象選択区間を用いて，同時事象が発生しないモデル上で振舞い仕様を満たしていることを示した．一方，図 8 に示した同時の事象に対応した状態遷移機械を追加したシステムでは，同時の事象が発生した場合の検証を示した．

ところで，図 8 を追加したモデルでは，同時の事象が発生しない図 7 で示した振舞いも検証する必要がある．この場合，5.2 節で示した検証式を，そのまま再利用することができる．これは，

- 同時の振舞いの定式化と一般化，および
- 同時の振舞いを環境に局所化

したことに起因する．対象となる設計を変更しても，環境が変わらない限り検証式を再利用することができる．

6.2 振舞い検証の指針

検証の対象が与えられて振舞い検証を行う場合，検証の仕様を後から考えることは困難な作業である．本稿で提案した同時性を考慮した振舞い検証の枠組みは，振舞い設計の指針を含意する．5 節で示した検証の流れが検証の指針となり得る．

(1) 環境の振舞い定義
(2) 同時性を考慮した振舞い仕様の定義
(3) 並行システムの振舞い設計
(4) 逐次振舞いの検証
(5) 同時振舞いの検証

仕様記述からトップダウンで設計を行うことが，全体の設計効率の向上につながると考えられる．また，逐次振舞いから同時振舞いに段階的な検証することが，有効な手順となりうる．

6.3 関連研究

ステートチャートや UML 等の図式表現を，モデル検査器を用いて検証する試みが行われている．Roscoe らはステートチャートをプロセス代数 CSP[9] に変換して，モデル検査器 FDR[4] を用いて競合状態（race conditions）などのエラーを検出している [8]．Hsiung らは組込みシステム向けの UML モデルからモデル検査を行うためのフレームワークを提供している [5]．これらは，汎用の検証の枠組みを与えているが，検証の基準となる仕様を記述する方法は提示されていない．

7. おわりに

本稿では，複数事象発生の同時性を考慮した，並行システム設計の振舞い検証支援を目的として，同時性の概念を定式化し，同時性を含意した振舞い仕様記述法と，同時性を考慮した振舞い検証の枠組みを提示した．

今後の課題として，並行システムの全体の振舞いを記述するための区間の合成演算子として，

- 並行演算子と
- 非決定的選択演算子

の拡張が挙げられる．

謝辞 本研究の一部は，JSPS 科研費 24500049，24220001，2014 年度南山大学パッヘ奨励金 I-A-2 の助成を受けて実施した．

参考文献

[1] M. Ben-Ari: Principles of Concurrent and Distributed Programming 2nd edition, Addison-Wesley (2006).
[2] Clarle, E., Grumberg, O. and Peled, D.: Model Checking, The MIT Press (1999).
[3] 土居範久: 相互排除問題, 岩波書店 (2011).
[4] FDR3, http://www.cs.ox.ac.uk/projects/fdr/.
[5] P.A. Hsiung, C.W. Lin, C.H. Tseng, T.V. Lee, J.M. Fu, and W.B. See: VARTAF: An Application Framework for the Design and Verification of Embedded Real-Time Software, IEEE Trans. on SE, 30, 10, pp. 656–674 (2004).
[6] 中島震：モデル検査法のソフトウェアデザイン検証への応用, コンピュータソフトウェア, Vol.23, No.2, pp. 72–86 (2006).
[7] A.W. Roscoe: The Theory and Practice of Concurrency, Prentice Hall (1998).
[8] A.W. Roscoe and Z. Wu: Verifying Statemate Statecharts Using CSP and FDR, Formal Methods and Software Engineering, Lecture Notes in Computer Science, 4260, pp. 324–341, Springer (2006).
[9] A.W. Roscoe: Understanding Concurrent Systems, Springer (2010).