並行システムの検証と実装

平成27年度シラバス

2015年1月9日

国立情報学研究所

トップエスイープロジェクト

代表者　本位田　真一

1. 科目名

並行システムの検証と実装

2. 担当者

初谷 久史

3. 本科目の目的

本科目の目的は

要求を満たす信頼性の高い並行システムを設計できるようになること

です．ここでいう設計とは次の3つの作業を意味します：

1. 並行システムの振る舞いを表すモデルを作成すること
2. 作成したモデルが与えられた要求や仕様を満たしていることを検証すること
3. 検証されたモデルに基づいてシステムを実装すること

この目的を達成するために，本科目では以下の4つのサブゴールを設定しています：

1. 並行システムについて学ぶ
2. 並行システムの理論である CSP 理論を学ぶ
3. 並行システムの振る舞いをモデル化し CSP 理論に基づいて検証するツールの使い方を学ぶ
4. 並行システムの実装方法について学ぶ

以下，4つのサブゴールについて説明します．

3.1. 並行システムについて学ぶ

並行システムとは何かをひとことでいうと次のようになります：

> 「複数の構成要素からなるシステムで，各構成要素が並行に動作し，互いに作用を及ぼしあいながら全体として目的の仕事をするシステムのこと」

設計する立場から並行システムを考えると，それ以外のシステムとは異なる点が大きく分けて2つあります．

相互作用と振る舞い

1つはシステムの構成要素がお互いに影響を及ぼしあうことです．これを**相互作用**といいます．一般には構成要素間だけでなくシステムそれ自身もユーザや他のシステムと相互作用をします．システムを外から観た場合，システムの**振る舞い**とはどのような相互作用をするかということであって，システム内部の作りや計算などには直接は関係しません．したがってシステムに対する要求やシステムの仕様は相互作用に基づいて記述することになります．同じことが相互作用をする構成要素に対してもいえます．並行システムの設計者はシステムや構成要素の振る舞いを相互作用に基づいて記述したり比較したりできる必要があります．入出力の関係としてとらえることのできるシステムでは主に計算ロジックの正しさが問題でしたが，並行システムでは内部計算ロジックの正しさに加えて相互作用としての振る舞いの正しさが問題になるということです．

並行性から生じる問題

並行システムがそれ以外のシステムとは異なるもう1つの点は，構成要素が並行に動作するというまさにその点から出てくる特徴的な性質があることです．厳密にいえばこれらは必ずしも並行システムに特有のものというわけではありませんが，並行システムではよく表れる性質であり，設計において考慮が必要となります．

性質は4つあります：

1. 状態数の組み合わせ爆発
2. 非決定性
3. デッドロック
4. ライブロック（発散）

状態数の組み合わせ爆発

状態数がそれぞれ m, n である構成要素2つからなるシステムの全状態数は，相互作用による制約がなければ $m \times n$ になります．このことからわかるようにシステムの状態数は構成要素の数に対して指数関数的に増えていきます．これを状態数の組み合わせ爆発，あるいは単に状態爆発といいます．

莫大な数の状態があるということは設計段階での考慮やテストによる網羅的な検証が難

しくなるということを意味します.

非決定性

2つ以上の構成要素が並行に動作する場合，全体としての処理の実行順序に任意性があります．その結果としてシステムが同じ状態にあり，かつ外界も同じ状況にあるにも関わらず，実行のたびに振る舞いが異なるという性質が現れます．システムが持つこのような性質のことを**非決定性**といいます.

非決定性はシステムの検証に重大な影響を及ぼします．同じ状況設定をしても振る舞いが異なる場合があるのですから，1度のテストが成功しても検証できたとはいえません．非決定性は確率的な現象ではないこともあるので，テストの回数を増やしても信頼性を高めることができない場合があります．たとえばテスト環境で1万回テストして発生しない問題が，リリース環境では100%発生するなどということがありえます.

デッドロック

2つ以上の構成要素が相互作用を行う場合，応答待ちの要求が循環してどれも動けなくなるという現象が起こりえます．これを**デッドロック**といいます.

ライブロック（発散）

2つ以上の構成要素が進捗のない相互作用を繰り返し続ける状況が発生し，結果としてそれ以外の構成要素やシステム外部からの要求を受け付けなくなるという現象が起こりえます．これを**ライブロック**または**発散**（divergence）といいます.

§

以上のように並行システムには他のシステムとは異なる特徴があります．要求を満たす信頼性の高い並行システムを設計できるようになるためには，これらの特徴について学ぶ必要があります.

3.2. 並行システムの理論である CSP を学ぶ

並行システムの設計を行うにあたっては，計算とは別に相互作用に基づいて振る舞いを記述したり比較したりするという新しい側面と，非決定性やデッドロックのように問題と

なりうる性質とに対処するための理論が必要になります．ここでいう理論の役割は文章問題に対する方程式，あるいは物理学の問題に対する微分方程式のようなものです．問題を記述し，解を導出し，結果を吟味するための理論です．

並行システムを記述し，その性質を議論するための理論には，オートマトン，ペトリネット，プロセス代数などさまざまなものがあります．本科目ではプロセス代数の中の1つである CSP (Communicating Sequential Processes) を採用します．

CSP 理論を学ぶと次のことができるようになります：

1. 並行システムやその構成要素の振る舞いを相互作用に基づいて記述することができるようになります．
2. 2つの振る舞いを相互作用に基づいて比較できるようになります．典型的には仕様とシステムの振る舞いを比較することができるようになります．
3. 前項で説明した問題となりうる非決定性等の性質に対して，理論的に対処することができるようになります．具体的には非決定性があっても問題がないことを論証したり，デッドロックが存在しないことを証明したりすることです．

2番目の項目について補足をします．「仕様とシステムの振る舞いを相互作用に基づいて比較する」ということは自明なことではありません．第1にシステムの振る舞いが仕様を満たしているということは，仕様で規定された振る舞いとシステムの振る舞いが同じであるという意味ではありません．一般に仕様では許容範囲としての意図的な選択の幅が用意されており，システムの実装ではその範囲から選択を行うことになります．したがって仕様と実装の振る舞いの関係は対称ではありません．CSP 理論ではこの関係が数学的に厳密に定義されており，**詳細化関係**といいます．

第2に比較をどのような基準で行うかという点でいくつかの選択肢があります．言い換えると詳細化関係にはいくつか種類があるということです．本科目ではこれらのうちで応用上重要な2つの基準について解説します．1つはシステムの振る舞いを外から観たときに観測できる相互作用の系列に基づいて比較を行う方法で，**トレース**方式[1]といいます．この基準を使うと，システムの振る舞いが仕様で規定されている振る舞いの範囲に収まっているかどうかを確認することができます．平たくいえば「仕様がやってもよいということだ

[1]参考文献1「並行システムの検証と実装」における用語に従いました．p.186 脚注3)参照

けをやっているかどうか」を確認できます．システムがもつこの性質を**安全性**（safety）といいます．トレース方式では安全性の確認ができるということです．

もう1つは非決定性まで考慮した，より精密な基準で比較を行う方法で，**安定失敗**方式といいます．この基準を使うと「仕様がやらなければならないということをやっているかどうか」を確認できます．システムが持つこの性質を**活性**（liveness）といいます．安定失敗方式では安全性に加えて活性の確認ができるようになります．

まとめると，CSP 理論を学ぶことによって，並行システムを開発する上で重要な概念である詳細化関係および安全性・活性という性質と，それを確認するための2つの方式であるトレース方式・安定失敗方式を習得することができます．

3.3. 並行システムの振る舞いをモデル化し CSP 理論に基づいて検証するツールの使い方を学ぶ

CSP 理論があれば原理的には設計の問題を机上で解決することができます．しかし現実のシステム設計では作業の量が膨大ですから，すべてを手作業で実行することは困難です．量が多ければ手作業では間違いを犯す可能性もあります．

加えて CSP 理論は数学的な理論なので，形式的な記号操作に慣れていない人にとってはシステム開発とのギャップがあり，とっつきにくいところがあります．

この2つの課題を解決するために本科目では SyncStitch というツールを使います．SyncStitch は CSP 理論に基づいたツールで，以下のことができます：

1. システムや構成要素の振る舞いを状態遷移図として記述することができます．
2. CSP 理論に基づくトレース方式・安定失敗方式の2つの方式で振る舞いを比較することができます．
3. デッドロックやライブロックがあるかどうかを調べる機能があります．

本科目では，一般によく知られた状態遷移図という表現形式を用いて，ツールを使いながら CSP 理論の概念を学んでいくというアプローチを採ります．そして概念を習得できた後で，CSP 理論本来の形を解説し，より理解を深めてもらいます．

3.4. 並行システムの実装方法について学ぶ

本科目では並行システムを実装する3つの方法について説明します：

1. CSP ライブラリを使った実装
2. 同期プリミティブを使った実装
3. 不可分操作を使った実装

CSP ライブラリを使った実装

CSP 理論に基づいて設計した並行システムを実装するもっとも直接的な方法は，CSP をサポートするプログラミング言語やライブラリを使うことです．この方法の利点は考え方が1つに統一できること，設計モデルから実装への変換がほぼ機械的であり容易であること，対応関係が明確なので実装の際に問題が混入する可能性が少ないことです．欠点としては，既存システムとの関係やメンバのスキルが制約となって導入できない場合があること，パフォーマンスが問題になることがある，などです．現在利用できるプログラミング言語処理系／ライブラリを付録の表に示しました．

本科目では MCCSP という C 言語用の CSP ライブラリを使います．このライブラリの特徴はシンプルな実装で内部を理解しやすく，改造して実験できるなど教育向けに適しているという点にあります．欠点はジャイアントロックを使ったナイーブな実装なのでパフォーマンスが悪いことです．しかし構成が簡単な MCCSP で1度実装方法を習得すれば，より高速で高度な機能を持つライブラリにステップアップすることはそれほど難しくないので，効率のよい学習順序であると考えます．

同期プリミティブを使った実装

共有メモリ型のアーキテクチャ上でオペレーティングシステムが提供するミューテックスやセマフォといった同期プリミティブを使って並行システムを実装するという手法は広く使われています．したがってそのような実装方法においても信頼性の高いシステムが構築できるようにすることが重要です．そこで，同期プリミティブの振る舞いを CSP でモデル化し，それを使ってシステムの検証を行った後に実装をするという方法を解説します．

不可分操作を使った実装

マルチコア／メニーコア時代に入り，システムの並列性を高めて全体のパフォーマンスを上げる技術は今後ますます重要になります．そのためのプリミティブとして Test and Set (TAS), Compare and Swap (CAS), Transactional Memory などのいわゆる不可分操作（アトミック操作）の利用が重要になります．その応用の1つとしてロックフリーアルゴリズムを例にとり，不可分操作をモデル化し，検証してから実装する方法について解説します.

4. 本科目のオリジナリティ

理論，ツール，実装技術の3点から本科目のオリジナリティを説明します.

理論

本科目では並行システムについて議論するための基礎となる理論としてプロセス代数の1つである CSP を採用します.

まず，プロセス代数を採用する理由は，並行システムでは相互作用が中心的な概念になるからです．他の手法では状態論的な振る舞いに力点が置かれており，システムと外界の間，およびシステム内部の構成要素間で行われる相互作用を議論する場合にあまり適切ではないと考えるからです.

次に，数あるプロセス代数の中で CSP を選択した理由は大きく分けて2つあります.

1つは理論の中で振る舞いについての詳細化関係が定義されていることです．仕様と実装という特定の2者の間では正当性関係と同じ意味になります．大規模なシステム開発では仕様から実装へ直接進むことはまれで，段階を追って徐々にシステムの詳細を詰めていくことになります．その際，中間の設計成果物を作る段階で，それ以前の成果物と矛盾していないかどうかを確認する必要があります．それが詳細化関係です．CSP はこの詳細化の概念によって，いわゆる段階的詳細化（Stepwise Refinement）を支援します．他の手法との比較でいえば，B-Method が状態論的な詳細化の概念に基づいて段階的詳細化を支援しているのと同じように，相互作用的な詳細化の概念に基づいて段階的詳細化を支援しているということです.

もう1つは CSP が多様な意味論を持っているという点です．まずプロセス代数としての代数的意味論と計算系としての操作的意味論があり，どちらも検証の技術，特にツールでの自動検証の基礎として重要な役割を果たしています．そしてもっとも CSP に特徴的なことは複数の表示的意味論を持つことです．その中でもトレース方式と安定失敗方式は安全性（safety），活性（liveness）といった設計上重要な概念に対応しており，並行システムの理解だけでなくツールによる効率の良い検証の基礎ともなっています．

以上のように，本科目では CSP を採用したことにより並行システムに関する重要な概念と実践的な技術を学ぶことができます．

ツール

本科目のオリジナリティの2つ目は，状態遷移図というよく知られた表現形式を使って CSP に入門できる点です．ツールのサポートがあるので，状態遷移図で記述した振る舞いを CSP に変換し検証することができます．加えて検証の結果も視覚的な計算木を対話的に操作しながら分析することができるので，直感的に理解しやすくなっています．

実装

実装面でのオリジナリティは2つあります．1つはソースコードで提供された内部理解が容易な CSP ライブラリを使用する点です．これによってモデルから実装への距離が短くなるだけでなく，ライブラリ自体が CSP の理解および並行システムの実装例として役に立ちます．もう1点は不可分操作に基づくロックフリーアルゴリズムを題材に取り上げるという点です．不可分操作を CSP でモデル化し検証する事例を学ぶことで，信頼性が高く性能の良い並列システムを構築する技術が身に付きます．

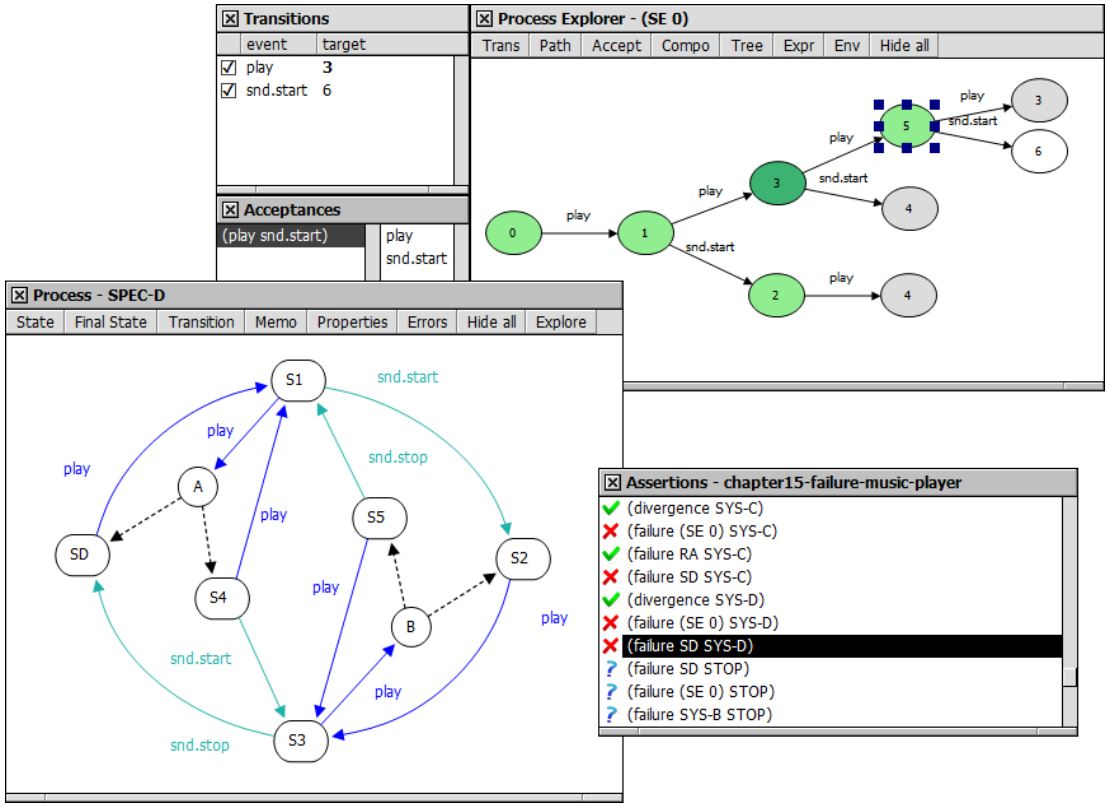

図1. SyncStitch のスクリーンショット

5. 本科目で扱う難しさ

すでに述べたように，並行システムには状態数の組み合わせ爆発，非決定性，デッドロック，ライブロックといった特徴的な性質があり，それらが信頼性の高いシステム設計の難しさにつながっています．特に非決定性とバグが組み合わさった場合は，まれに発生する，発生する頻度が予測できない，特定のタイミングで発生することがある，といった難しい問題になります．バグ発生の結果がデッドロックやライブロックにつながるという場合すらあります．

これらの問題はテストによる再現だけで分析したり，検証したりすることはできません．これらの問題を扱うことのできる CSP 理論に基づいてモデル化し，ツールの力を援用して設計段階で検証する必要があります．

6. 本科目で習得する技術

1. CSP 理論に基づく並行システムの振る舞いの記述と検証の技術
   a) 相互作用に基づく振る舞いの記述
   b) 振る舞いの検証
      i. 代数的意味論に基づく振る舞い表現の変換と比較
      ii. 操作的意味論に基づく遷移の導出
      iii. 表示的意味論に基づく詳細化関係の検証
2. ツール
   a) 状態遷移図とプロセス式（CSP）による振る舞いのモデル化
   b) シミュレーションによる振る舞いの確認・分析
   c) デッドロック検査
   d) ライブロック検査
   e) トレース方式による詳細化検査，安全性の検査
   f) 安定失敗方式による詳細化検査，活性の検査
3. 実装
   a) C 言語用 CSP ライブラリ MCCSP による並行システムの実装
   b) 同期プリミティブによる並行システムの実装
   c) 不可分操作による並行システムの実装

7. 前提知識

C 言語によるプログラミングの知識を前提とします．実装の回および課題レポートでは pthread および POSIX Semaphore を使用した実装を行います．pthread および POSIX Semaphore については講義の中で軽く解説をしますが，簡単なプログラムを動かした経験があるとスムーズに作業ができると思われます．

ツールの使い方については0から説明しますので，前提知識は不要です．ツール上で作成するモデルの中で，計算をする部分ではプログラミング言語 Scheme を使いますが，これについても講義の中で解説をします．

CSP 理論の前提知識は不要です．CSP 理論を解説する回では集合と論理の記号を使用します．基礎理論で解説されている程度の知識を前提とします．

8. 講義計画

概要

第1, 2回： 並行システム概説，状態遷移図による逐次プロセスのモデル化

第3, 4回： 式による逐次プロセスのモデル化，並行合成，デッドロック，逐次合成

第5, 6回： プロセスの動的生成と終了，隠蔽，非決定性，発散

第7, 8回： トレース方式による詳細化検査

第9, 10回： 安定失敗方式による詳細化検査

第11, 12回： CSP 理論

第13, 14回： CSP ライブラリによる実装，同期プリミティブによる実装

第15回： 不可分操作によるロックフリーアルゴリズムの実装

詳細

第1, 2回： 並行システム概説，状態遷移図による逐次プロセスのモデル化

1. 並行システム概説
2. ツール SyncStitch 概要
3. プログラミング言語 Scheme 概説
4. チャネル通信
5. 状態変数
6. ガード

第3, 4回： 式による逐次プロセスのモデル化，並行合成，デッドロック，逐次合成

1.プロセスの終了

2.式によるプロセスの記述

3.並行合成

4.デッドロック検査

5.逐次合成

6.演習

7. レポート課題解説

第5, 6回： プロセスの動的生成と終了，隠蔽，非決定性，発散

1.プロセスの動的生成と終了

2.隠蔽

3.非決定性

4.発散検査

5.演習

6.レポート課題解説

第7, 8回： トレース方式による詳細化検査

1.トレース

2.安全性

3.トレース方式による詳細化検査

4.演習

5.レポート課題解説

第9, 10回：安定失敗方式による詳細化検査

1.拒否

2.トレース後のプロセス

3.安定失敗

4.活性

5.安定失敗方式による詳細化検査

6.演習

7.レポート課題解説

第11, 12回： CSP 理論

1.シンタックス

2.表示的意味論

a) トレース

b) 安定失敗

3.代数的意味論

4.操作的意味論

5.演習

6.レポート課題解説

第13, 14回： CSP ライブラリによる実装，同期プリミティブによる実装

1. CSP ライブラリ MCCSP による実装

a) MCCSP ライブラリ概説

b) 事例：スケジューラ解説

c) モデル化と検証

d) 実装

2. 同期プリミティブによる実装

a) pthread 概説

b) 事例：リングバッファ

c) pthread のモデル化

d) リングバッファのモデル化と検査

e) 実装

3. レポート課題解説

第15回： 不可分操作によるロックフリーアルゴリズムの実装

1. Compare and Swap (CAS) 解説

a) 事例：ロックフリースタック

b) モデル化と検査

c) ABA 問題

d) 実装

2. まとめ

9. 教育効果

本科目を受講することにより，プロセス代数 CSP に基づくモデル化，検証，実装の方法を習得できます．理論から実装までの流れを理解することによって，検証された信頼性の高い並行システムを開発できるようになります.

10. 使用ツール

1. SyncStitch

   http://www.principia-m.com/jp/syncstitch.html

2. MCCSP

   講義までに配布予定

3. Cygwin

   https://www.cygwin.com/

11. 実験及び演習

モデル化と検証では，実際にツールを使い，手を動かしてもらうことで効率よく学習できます．検査の結果を対話的に操作しながら理解する努力をすることで，並行プロセスに特徴的な問題の理解力・分析力を高めることができます．

モデルだけにとどまらず実装まで行うことによって，非決定性やデッドロックといった現象が実際に発生する様子を観察することができます．これによりテストだけでは検証が難しいということを体感することができます．最後に検証の済んだモデルを実装することで，検証の有効性を実感することができます．

12. 評価

課題レポートおよび出席日数を総合して評価します．

13. 教科書／参考書

- 磯部祥尚（著），東野輝夫（監修），並行システムの検証と実装 — 形式手法 CSP に基づく高信頼並行システム開発入門，トップエスイーシリーズ 実践講座 6 ,近代科学社, 2012. https://sites.google.com/site/topsevic/home
- C. A. R. Hoare, Communicating Sequential Processes, 1985.
  http://www.usingcsp.com/ からダウンロード可能.
  邦訳: ホーア CSP モデルの理論, 吉田 信博(翻訳), 丸善, 1992
- A. W. Roscoe, The Theory and Practice of Concurrency. Prentice Hall, 1998.
  http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/publications/68b.pdf からダウンロード可能.

付録

表1　CSP ライブラリとプログラミング言語

言語	ライブラリ	関連 URL（ダウンロード可能）
Java	JCSP	http://www.cs.kent.ac.uk/projects/ofa/jcsp/
C++	C++CSP	http://www.cs.kent.ac.uk/projects/ofa/c++csp/
Jibu	C# (.NET)	https://github.com/pascalvancauwenberghe/Jibu
Haskell	CHP	http://www.cs.kent.ac.uk/projects/ofa/chp/
Python	PyCSP	http://code.google.com/p/pycsp/
Python	Python-CSP	http://code.google.com/p/python-csp/
Go 言語 (Google)		http://golang.org/
XC (XMOS)		http://www.xmos.com/jp