並行システムの検証と実装

執筆中（2011 年 8 月 27 日版）

磯部祥尚（産業技術総合研究所, 国立情報学研究所）

まえがき

複数の人や物が協力（協調）しながら**並行**に振舞う機会は多い。例えばレストランでは、注文を受ける作業、料理をする作業、片づけをする作業等を複数の人が分担し、互いに連絡を取りながら食事を提供する業務を並行に行っている。このような作業を全て 1 人が逐次的に行った場合、サービスの低下と作業時間の増加は避けられない。同様にコンピュータの世界でも、複数の**プロセス**が並行に動作することによって、ユーザからの入力を受け付けつつ、内部で複雑な計算を行い、その計算過程を適時表示するような並行処理を実現している。これはユーザに対する利便性を高めるだけでなく、並列処理装置があれば処理時間の短縮も可能になる。**並行システム**とは、このような協調動作する複数のプロセスから構成されるシステムのことである。

最近（2011 年）ではマルチルコア CPU が多くのパソコンに搭載され、並列処理装置が身近なものになってきている。従来、パソコンの性能向上は主に CPU の周波数を上げることによって行われ、周波数が上がれば、どのようなプログラムの実行速度も改善されてきた。一方、マルチコア CPU のコア数を増やしても、プログラムが並行処理に対応していなければ、実行速度の改善は望めない。すなわち、マルチコア CPU の恩恵を受けるには、処理の並行化を欠かすことはできない。

しかし、相互作用をもつ複数のプロセスから構成される並行システムの動作を正確に予想することは難しく、デッドロックのような並行処理特有の不具合を内在させてしまう可能性がある。また、ネットワーク接続された複数のコンピュータで複数のプロセスを分散協調動作させる場合もある。このような分散システムではネットワークの断線や一部の機器の電源断など、非正常系の動作に対応する対策も講じる必要があり、その設計はさらに困難になる。

本書では、信頼性の高い並行システムを開発するための方法として、並行システムの理論、検証、実装について、特にそれらのつながりを重視して解説する。 尚、**並列処理**と**並行処理**の違いについて説明しておく。複数の処理装置（マルチコアやマルチプロセッサなど）を用いて複数の処理を実際に同時に実行することを**並列処理** という。これに対し、一つの処理装置を用いて複数の処理を時分割で実行することを**疑似並列処理**（見掛け上の並列処理）という。本書では**並行処理** という用語を、「複数の処理を時間的な重なりをもっ

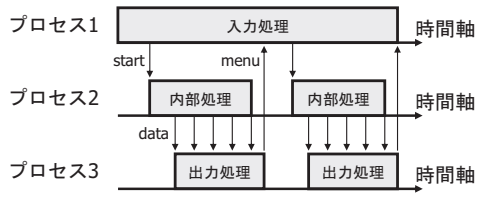

図 1　入力処理、内部処理、出力処理を並行に実行する動作の例

て（並行に）実行すること（ある処理を実行中に他の処理を実行すること）」という意味で用いる。すなわち、並行処理は並列処理と疑似並列処理の両方を含む、より広い意味の用語である。

並行システムの特長

並行システムの次の特長について説明する。

1. （並行処理）相互に関係する複数の処理を並行に実行することができる。
2. （高速処理）並列処理装置上に実装することによって処理速度を向上できる。

並行処理

並行システムは、複数の処理を並行に実行することができる。例えば、入力処理、内部処理、出力処理を実行する 3 つのプロセスをもつ並行システムでは、次のような複数の処理を並行に実行可能である。

1. 入力処理プロセスで常に入力を受け付けつつ、
2. 内部処理プロセスで入力操作に応じた内部処理を実行し、
3. 出力処理プロセスで内部処理中にその進捗状況を表示する。

その動作例を図 1 に示す。水平方向の時間軸上の長方形がプロセスの実行期間、上下の矢印がプロセス間の通信を表している。図中、入力処理に応じた内部処理開始を指示する通信（start）、内部処理の進捗状況を表示するために必要な通信（data）、表示内容によって入力メニューが変化することを伝える通信（menu）が、適時実行されている。

図 2 逐次システムによる 2 つの処理 A,B の逐次実行例

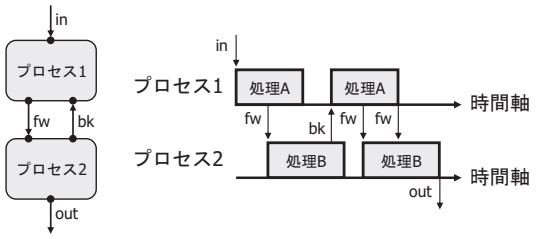

図 3 並行システムによる 2 つの処理 A,B の並列実行例

このような複数のプロセスが通信によって協調動作する並行処理は、ユーザに対する利便性を向上させるだけでなく、入出力処理部と内部処理部が分離している分散システムでは不可欠である。

高速処理

複数の処理を同時に実行できる並列処理装置上に並行システムを実装すると、各プロセスは実際に並列に処理されるため、その全体の処理速度を向上することができる。並列処理装置上に**逐次システム**（プロセスが 1 つのシステム）を実装しても、一度に実行できる処理は 1 つであるため、その並列処理能力を活かすことはできない。CPU のマルチコア化が急速に進んでいる現在、並行システムを構築することは非常に有意義である。

例えば、互いに依存関係があり、同時に実行する必要のない 2 つの処理 A,B があるとする。このとき、図 2 のように処理 A,B を逐次的に実行しても問題はない。しかし、もし 2 つの処理 A,B の一部でも独立に実行可能であるならば、図 3 のように並列処理することによって処理時間の短縮が期待できる（実際には通信等の時間も考慮する必要がある）。

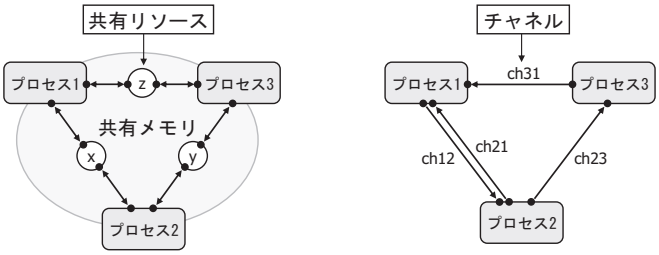

図 4　共有メモリ通信（左）とメッセージパッシング通信（右）の構成例

本書の目的

並行システムの通信方式は、共有メモリ上でメッセージを交換する**共有メモリ通信方式**と、チャネルを通してメッセージを送受信する**メッセージパッシング通信方式**に大別できる（図 4 参照）。一般に、共有メモリ通信はメッセージパッシング通信に比べて高速であるが、その動作を的確に予測することは難しく、**デッドロック**や**リソース競合**などの不具合が発生する可能性が比較的高い。

本書では、信頼性の高い並行システムを構築することを目的として、次の 3 つのキーワード（CSP[6, 22], FDR[14], JCSP[18]）を軸に、並行システムのモデル化、検証、実装の方法について解説する。

CSP ：並行システムを記述・解析するための理論　　（プロセス代数）
FDR ：CSP 理論に基づく並行システム検証ツール　　（モデル検査器）
JCSP ：CSP のプロセスモデルの実装ツール　　（Java ライブラリ）

JCSP は Java で**同期型**（**ランデブー**とも呼ばれる）のメッセージパッシング通信（CSP モデルの通信方式）を実装するためのライブラリである。同期型通信とは、送信と受信が同時に実行される通信方式のことである。例えば、送信側のプロセスが送信できる状態になっても、受信側のプロセスがまだ受信できない状態にあるときは、受信できる状態になるまで送信も延期される。受信側の状態に関係なく送信される**非同期型**のメッセージパッシング通信方式と比較して、同期型を使うと並行システム全体の動作が理解しやすくなり、またメッセージの取りこぼしがないので、より信頼性の高い並行システムを構築すること

ができる。さらに信頼性を高めるために、モデル検査器 FDR によって並行システムの動作を検証することも可能である。尚、本書では次の用語を用いる[1)]。

CSP モデル ：並行システムの CSP 記述（形式的な記述）
FDR スクリプト ：モデル検査器 FDR で読込み可能な記述（FDR 入力言語）
JCSP プログラム ：ライブラリ JCSP を利用している Java プログラム

本書の使い方

本書では、並行システムの CSP によるモデル化技術、モデル検査器 FDR による検証技術、JCSP による実装技術を解説する。モデル検査器 FDR の検証は完全に自動化されており、クリック一つで使うことができるが、FDR の検証機能をより適切に使うため、また検証結果が偽の場合の反例を正しく理解するためには、CSP 理論の知識が必要である。本書では、FDR を使うために必要になる CSP 理論についても説明している。

本書を読む前に必要な知識はクラスやオブジェクトなどの Java プログラムの基礎知識程度である。並行プログラムやプロセス代数についての予備知識は必要としてない。本書は次のような目的をもっている人に読んでほしい。

- **プロセス代数を初めて勉強する人**：本書では、プロセス代数がどのような理論で、どのように役に立つのかを説明している。プロセス代数は、並行システムの動作を**式**の形で記述し（構文）、その動作を状態遷移図に変換し（意味）、2 つの動作（例：並行システムと逐次システム）の間の等しさなどを判定する（検証）ための理論である。さらに詳しく CSP 理論を勉強したい人は、文献 [6, 22] を参照してほしい。
- **並行システムの検証に興味をもっている人**：最近、SMV[17], UPPAAL[2], SPIN[8] などのモデル検査器がシステム検証に適用されるようになってきている。これらは並行システムと要求の記述に各々オートマトン（またはクリプキ構造）と時相論理を使っている。これに対し本書で紹介する FDR は並行システムも要求（仕様）もプロセス代数 CSP で記述する。本書は、初めてモデル検証器を使う人だけでなく、他のモデル検査器を使ったことのある人にも読んでほしい。

[1)] 基本的に FDR のプロセス記述は CSP モデルと同じであるが、FDR スクリプトは検証対象の CSP モデルの他に、データに関する記述や検証の表明等の情報も含んでいる。

- **並行プログラミングを始めたい人**：最近はマルチコア CPU の普及もあり、並行プログラミングの機会も増えてきている。是非、CSP モデルのチャネルを使った並行プログラミングの面白さを体験してほしい。尚、本書で紹介する JCSP の他にも、表 1 に示すように様々な CSP ライブラリが公開されており、本書の開発手法は Java 以外のプログラミング言語にも有効である。 尚、表 1 の Go 言語は Google が 2009 年秋に発表したプログラミング言語であり、CSP に基づく並列処理記述が言語レベルで可能になっている。また、XC は XMOS 社 [30] の組込み用イベント駆動型マルチコアプロセッサの実装言語であり、CSP はハードウェアの設計にも有効である。

表 1　様々な CSP ライブラリ/言語と入手先

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

本書で紹介するツールはアカデミック目的であれば全て無料でインターネットからダウンロードしインストールすることができる。付録でインストール方法を説明しているので、是非ツールを使いながら本書を読み進めてほしい。実際にモデル検査器 FDR で検証し、Java ライブラリ JCSP で実装することによって、検証結果どおりに動作することが確認できるはずである。

本書の構成

本書は、図 5 に示すように、CSP 理論→ FDR 検証→ JCSP 実装の流れを重視することを特徴の一つにしている。そこで、図 6 に示すように、本書はその流れ（CSP → FDR → JCSP）を繰り返すように構成されている。すなわち、CSP 理論、FDR 検証、JCSP 実装を別々に理解していくのではなく、それらの関係に常に気を配りながら、CSP, FDR, JCSP をまとめて理解できるようになっている。

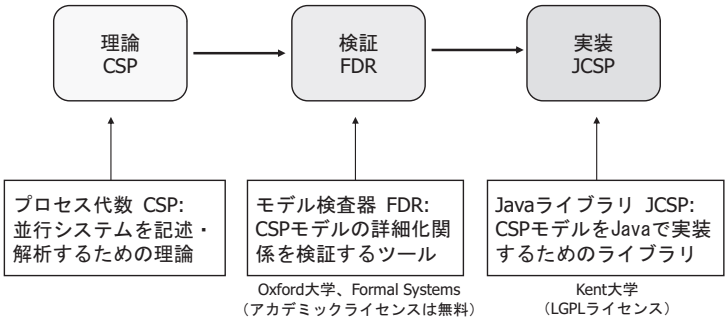

図 5　CSP 理論→ FDR 検証→ JCSP 実装の流れ

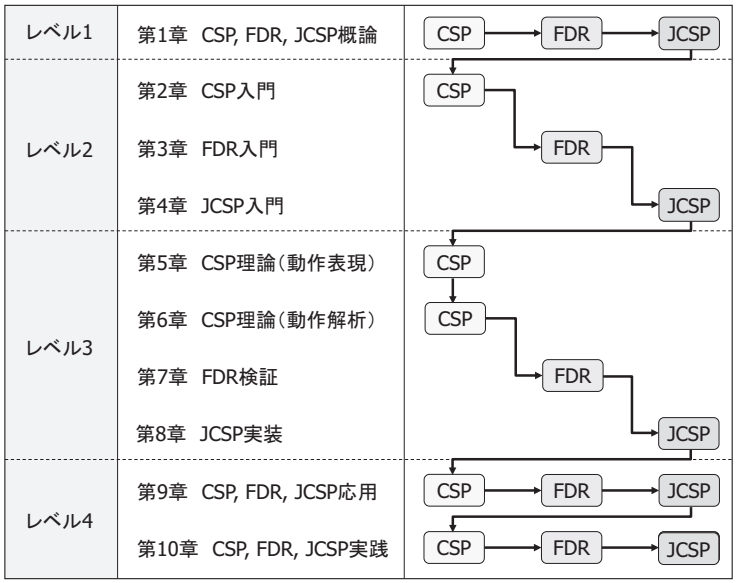

図 6　本書の章構成と学習レベル

本書はその学習レベルによって大きく 4 つに分かれている（図 6 参照）。レベル 1 で CSP → FDR → JCSP の流れを理解し、レベル 2 で CSP によるモデル化、FDR による検証、JCSP による実装の基本を学習する。レベル 3 では CSP による動作表現と解析方法を理解し、FDR による検証/デバッグ方法を習得すると共に、JCSP による並行/分散プログラミング技法を習得する。最後に、レベル 4 では CSP に基づくさらなる検証/実装技術を学習し、例題を用いて実習する。次に各レベルでの学習内容を説明する。

- **レベル** 1（第 1 章）：CSP, FDR, JCSP を使うことによって、どのようなことができるかを知る。また、簡単な並行システムの例を用いて、並行システムの CSP によるモデル化、FDR による検証、JCSP による実装という一連の流れを理解する。
- **レベル** 2（第 2 章〜第 4 章）：CSP でモデル化するために必要な並行プロセスの構文と、FDR で検証するために必要な補足的な構文（データや検証式等）を学習する。さらに、CSP モデルを JCSP プログラムに変換する方法を学習する。
- **レベル** 3（第 5 章〜第 8 章）：CSP モデルの動作を状態遷移図で表現する方法を学習すると共に、2 つの CSP モデルの等価性や詳細化関係の意味を理解し、それらの関係を FDR で検証する方法とデバッガの使い方を習得する。さらに、実装に便利な JCSP の機能やネットワークプログラミングについて学習する。
- **レベル** 4（第 9 章, 第 10 章）：より高度な検証方法と実装方法を学習し、実例を用いて並行システムをモデル化し、検証し、実装するための技術を実習する。

参考文献

[1] Philip Armstrong, Gavin Lowe, Tomasz Mazur, Joel Ouaknine, and Bill Roscoe. Csp model checking: New technology and techniques. http://www.comlab.ox.ac.uk/projects/cspmodelchecking/.

[2] G. Behrmann, A. David, and K. G. Larsen. *A Tutorial on Uppaal*. 2004. http://www.uppaal.com/.

[3] J. Bowen. The occam archive. http://vl.fmnet.info/occam/.

[4] Ronald S. Cok. *Parallel Programs for the Transputer*. Prentices-Hall, 1991. 共立出版より日本語訳「Transputer/occam による並列プログラミング入門」が出版されている.

[5] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, Vol. 18, No. 8, pp. 453–457, 1975.

[6] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[7] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. 本書は http://www.usingcsp.com/cspbook.pdf よりダウンロード可能。また、丸善より日本語訳が出版されている： C.A.R. ホーア (著), 吉田 信博 (訳) ホーア CSP モデルの理論, 丸善, 2002.

[8] G. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2004.

[9] Maven Hub. The jcsp library - jcsp-1.1-rc5. http://mavenhub.com/mvn/central/org.codehaus.jcsp/jcsp/1.1-rc5.

[10] Y. Isobe and M. Roggenbach. Webpage on Csp-Prover. http://staff.aist.go.jp/y-isobe/CSP-Prover/CSP-Prover.html.

[11] Y. Isobe and M. Roggenbach. A generic theorem prover of CSP refinement. In Nicolas Halbwachs and Lenore D. Zuck, editors, *TACAS 2005*, LNCS 3440, pp. 108–123. Springer, 2005.

[12] Y. Isobe and M. Roggenbach. A complete axiomatic semantics for the csp stable-failures model. In Christel Baier and Holger Hermanns, editors, *CONCUR 2005*, LNCS 4137, pp. 158–172. Springer, 2006.

[13] Jonathan Lawrence. Practical application of csp and fdr to software design. In A.E. Abdallah, C. B. Jones, and J. W. Sanders, editors, *Communicating Sequential Processes – The First 25 Years*, LNCS 3525, pp. 151–174. Springer, 2004.

[14] Formal Systems (Europe) Limited. Failures-divergence refinement: FDR2. http://www.fsel.com/.

[15] Formal Systems (Europe) Limited. Failures-divergence refinement: Fdr2 user manual. http://www.fsel.com/documentation/fdr2/fdr2manual.pdf.

[16] Kazuto Matsui. Prominent network. http://members.jcom.home.ne.jp/1355/.

[17] K. L. McMillan. *Symbolic Model Checking*. Springer, 1993.

[18] Peter Welch (University of Kent). Communicating sequential processes for java (jcsp). http://www.cs.kent.ac.uk/projects/ofa/jcsp/.

[19] University of Kent. Jcsp networking – some tutorial examples. http://projects.cs.kent.ac.uk/projects/jcsp/svn/jcsp/trunk/JCSP Networking.pdf.

[20] National University of Singapore. PAT: Process analysis toolkit. http://www.comp.nus.edu.sg/~pat/.

[21] Jan Peleska. Applied formal methods – from csp to executable hybrid specifications. In A.E. Abdallah, C. B. Jones, and J. W. Sanders, editors, *Communicating Sequential Processes – The First 25 Years*, LNCS 3525, pp. 293–320. Springer, 2004.

[22] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998. 本書は http://web.comlab.ox.ac.uk/oucl/work/bill.roscoe/pubs.html (No.68) よりダウンロード可能.

[23] A. W. Roscoe, Paul H. B. Gardiner, Michael Goldsmith, J. R. Hulance, D. M. Jackson, and J. B. Scattergood. Hierarchical compression for model-checking csp or how to check 10^{20} dining philosophers for deadlock. In *TACAS 1995*, LNCS 1019, pp. 133–152, 1995.

[24] Mark Russinovich. *Process Explorer v11.21*. 2008. http://www.microsoft.com/technet/sysinternals/.

[25] P. Ryan, S. Schneider, M. Goldsmith, G. Lowe, and B. Roscoe. *The Modelling and Analysis of Security Protocols: the CSP Approach*. Addison-Wesley, 2001.

[26] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[27] Steve Schneider. *Concurrent and Real Time Systems: The CSP Approach*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1999.

[28] Steve Schneider and Rob Delicata. Verifying security protocols: An application of csp. In A.E. Abdallah, C. B. Jones, and J. W. Sanders, editors, *Communicating Sequential Processes – The First 25 Years*, LNCS 3525, pp. 244–263. Springer, 2004.

[29] P. Welch, N. Brown, J. Moores, K. Chalmers, and B. Sputh. Integrating and extending jcsp. In *Communicating Process Architectures 2007*, pp. 349–370. IOS Press, 2007. http://www.cs.kent.ac.uk/projects/ofa/jcsp/jcsp-paper-2007.pdf.

[30] XMOS. イベント駆動型マルチコアプロセッサ. http://www.xmos.com/jp.

[31] 磯部祥尚. トップエスイー講座「並行システムの検証と実装」. http://staff.aist.go.jp/y-isobe/topse/vic/.

[32] 吉岡信和, 青木利晃, 田原康之. SPIN による設計モデル検証—モデル検査の実践ソフトウェア検証 (トップエスイー実践講座). 近代科学社, 2008.

[33] 中島震. SPIN モデル検査 – 検証モデリング技法. 近代科学社, 2008.