Vol. 24 No. 5　　情報処理学会論文誌　　Sep. 1983

臨界領域法に基づく並行プロセスシステムのソフトウェア設計手法†

広田豊彦††　大野豊†††

いくつかのプロセスが共有変数へアクセスするような並行プロセスシステムを設計する際には，各プロセスごとの動作の正当性，すなわちセマンティックインテグリティのみならず，共有変数へのアクセスの相互排斥，プロセス間の同期，デッドロックの防止などの問題に直面する．筆者らは，各プロセスの共有変数へのアクセスに着目して設計をすすめ，適切な臨界領域を設定することによって，システムの正しい並行処理を保証するような設計手法として臨界領域法を開発した．臨界領域法では，各プロセスの処理を細分化した処理ステップと，処理ステップ間の順序関係を規定した処理要求記述に基づいて設計を行う．まず各処理ステップの共有変数アクセスを調べ，各プロセスに対して最適な実行手順の選択を行う．そして相互排斥，デッドロック防止，シリアライザビリティなどを考慮して，臨界領域を設定する．臨界領域法における設計手続きは，半順序集合上の操作として定式的に記述され，(1)出発点となる処理要求記述が正しい限り，設計結果は並行処理に関する機能的正当性が保証される，(2)性能面をあまり問題にしない場合には，機械的に設計手続きをすすめることができる，などの特徴をもつ．

1. はじめに

いくつかのプロセスが共有変数へアクセスするような並行プロセスシステムを設計する際には，設計者は共有変数へのアクセスの相互排斥，プロセス間の同期，デッドロックの防止[1]などに注意しなければならない．従来の設計手法，たとえば，Parnas のモジュール化[2]，複合設計[3]，データ抽象化[4]などでは，モジュール化の指針は与えられるが，上述の並行プロセスの問題を取り扱っていないため，設計者は経験的にそれらの問題に対処する必要があった．一方，Concurrent Pascal[5] をはじめとする各種の並行プログラミング言語は，その言語の文法の制約によって正当な並行処理を保証することを目的としている．しかしこれらの言語は，1変数の相互排斥など，比較的低レベルの並行処理の正当性は保証できるが，デッドロックの防止やプロセス間の同期などのより高いレベルの並行処理に関して，その正当性を保証するのは困難である．

筆者らはこれまでオンラインファイルの並行処理に関する研究[6),7)]を行ってきたが，その成果を並行プロセスシステムの設計へ応用することにより，正当な並行処理を保証する設計手法として，臨界領域法を確立した．本論文では，臨界領域法に従った設計手順について述べ，さらに，その正当性の背景として，臨界領域法の定式化について述べる．

† Software Design Procedure for Concurrent Process System Based on Critical Region Method by TOYOHIKO HIROTA (Educational Center for Information Processing, Kyoto University) and YUTAKA OHNO (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学情報処理教育センター
††† 京都大学工学部情報工学科

2. 臨界領域法

2.1 並行プロセスシステムの正当性

並行プロセスシステムは，

(1) デッドロックを起こさない

(2) システム内の各プロセスの処理結果が正しい

の2条件によって正当性が保証される．このうち条件(1)については十分な議論が行われてきたが[8]，条件(2)に関してはあまり明確ではない．ここでは処理環境とシリアライザビリティ[9]の二つの概念を導入して，条件(2)について考察する．

プロセスの処理環境とは，システムの一部分の状態の集合であり，プロセスの実行がシステムのある部分 S と時刻 T において密接に関係しているとき，その状態 $S(T)$ は処理環境の要素となる．たとえば，プロセス X が時刻 T_1 に変数 F_1 へアクセスし，時刻 T_2 に変数 F_2 へアクセスするとき，それらの変数の値 $F_1(T_1)$, $F_2(T_2)$ はプロセス X の処理環境の要素である．

並行プロセスシステム中の一つのプロセス（以後並行プロセスとよぶ）の処理環境が，順次処理下における同一プロセスの処理環境の一つと等価であるとき，前者をシリアライザブルな処理環境とよぶ．**図1**ではプロセス Y はプロセス X が変数 F_1 の処理を終えると

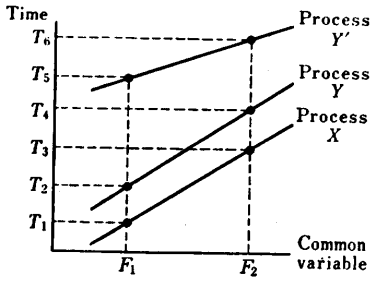

図 1 シリアライザブルな処理環境の例
Fig. 1 Example of serializable execution environment.

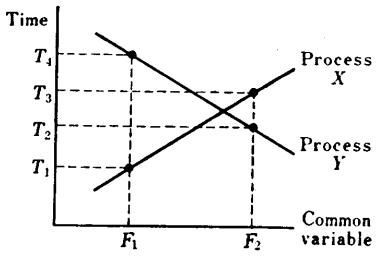

図 2 シリアライザブルでない処理環境の例
Fig. 2 Example of unserializable execution environment.

すぐにその変数へアクセスしている．他方，順次処理下では，Y は図中の Y' で示される形で実行される．このとき Y の処理環境 $\{F_1(T_2), F_2(T_4)\}$ と Y' の処理環境 $\{F_1(T_5), F_2(T_6)\}$ はいずれもプロセス X の処理後の状態であるという点で等価であり，それゆえに Y の処理環境はシリアライザブルである．なお**図 2** は X, Y の処理環境がともにシリアライザブルでない例を示している．

以上の議論から明らかなように，プロセスの順次処理下での動作の正当性（セマンティックインテグリティ[10]）を前提とすると，並行プロセスの処理結果の正しさ（前述の条件(2)）はシリアライザブルな処理環境によって保証することができる．

2.2 臨界領域法

設計者はまず対象システムの各プロセスの処理を，共有変数に着目して処理ステップに分割し，処理要求記述 (PRD) を構成する．

(1) 各処理ステップでは一つの共有変数のみへアクセスする．

(2) 処理ステップ間には半順序が規定されている．

条件(1)は臨界領域法が共有変数へのアクセスに基づいていることによる制約であり，条件(2)は，PRD においては処理ステップ間の順序のうちで必要不可欠なもののみを規定し，他は後の設計に委ねるための要請である．

[例 1] 三つのプロセス X_1, X_2, X_3 と六つの共有

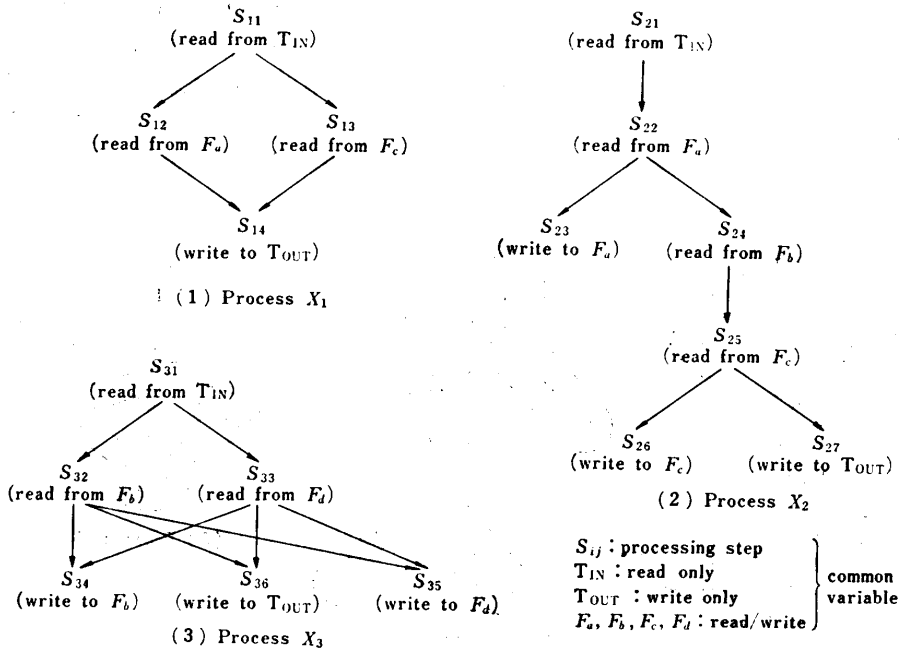

図 3 例題システムの処理要求記述
Fig. 3 Processing requirements description of the example system.

変数 T_{IN}, T_{OUT}, F_a, F_b, F_c, F_d から構成される例題システムについて，その PRD を優先度グラフの形で示したものが**図 3** である．

臨界領域法では PRD に基づいて実行手順の選択と臨界領域の設定の 2 段階で設計を行う．

2.2.1 実行手順の選択

PRD では各プロセス内の処理ステップ間の順序関係は半順序で規定されている．したがって実際にどのような順序で処理ステップを実行するかという実行手順に関しては，一般に選択の余地が残されている．そこで以下に述べる局所化と相似化の二つの手続きによって選択を行う．

共有変数へのアクセスは臨界領域内で排他的に行われなければならない．一方，システム全体の性能の立場からは，そのような排他的アクセスはなるべく短い時間内に終了することが望ましい．したがって同一共有変数へアクセスする処理ステップが複数個ある場合，それらを連続して実行するような実行手順を選択する．このための手続きを局所化とよび，以下のように行う．

〔局所化〕

（L 1） 処理ステップを共有変数に着目してグループ化する．

（L 2） 処理ステップ間の順序関係をグループ間の順序関係へ写像する．二つ以上のグループ間に双方向の順序関係が存在する場合には，全体を一つのグループとする．

（L 3） グループ間の順序関係を元の処理ステップ間の順序関係へ逆写像する．

この手続きにより，異なる共有変数 V_1, V_2 へアクセスするステップ S_a, S_b に対して順序対 $\langle S_a, S_b \rangle$ があり，同じく V_1, V_2 へアクセスするステップ S_x, S_y に対しては順序対がない場合，新たな順序対 $\langle S_x, S_y \rangle$ が追加される．

［例 2］ 例 1 のシステムに対する局所化の適用を**図 4** に示す．ここではプロセス X_2 と X_3 に対してステップ間順序対が追加されている．この結果，プロセス X_2 では

$$S_{21} \rightarrow S_{22} \rightarrow S_{23} \rightarrow S_{24} \rightarrow \cdots$$

という実行手順が決まり，F_a へアクセスする二つのステップ S_{22} と S_{23} は連続して実行される．

2.1 節で述べたシリアライザブルな処理環境の保持のためには，図 1 に示したように全プロセスが同一の順序で共有変数へアクセスすることが望ましい．そこで各プロセスの処理ステップにおける共有変数へのアクセスを解析し，なるべくアクセスの順序が同一になるように実行手順の選択を行う．このための手続きを相似化とよび，以下のように行う．

〔相似化〕

（S 1） 各プロセスの処理ステップ間の順序関係をすべて，共有変数間の順序関係へ写像する．

（S 2） 得られた順序関係中にループが存在する場合，ループに含まれるすべての順序対をその順序関係から取り除く．

（S 3） 共有変数間の順序関係を各プロセスの処理ステップ間の順序関係へ逆写像する．

注） （S 2）は，（S 3）における逆写像が元の半順序を破壊するのを避けるための手続きである．

［例 3］ **図 5**（1）は例 1 のシステムから得られる共有変数間の順序関係である．これを各プロセスの処理

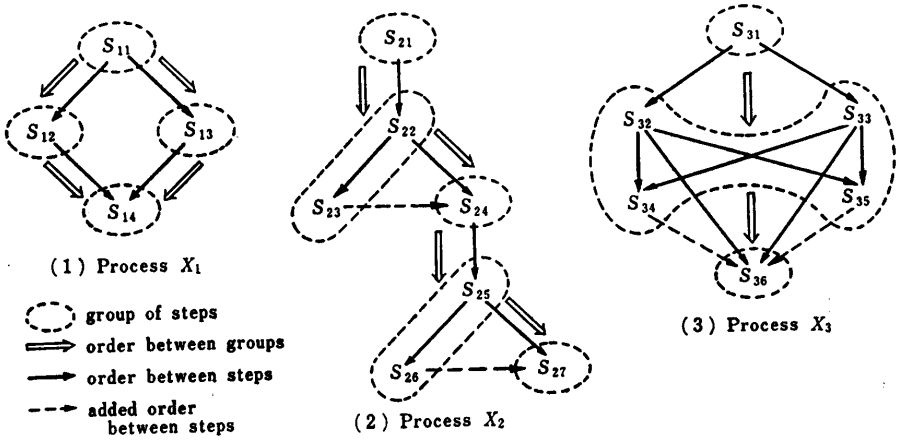

図 4 例題に対する局所化
Fig. 4 Localization for the example system.

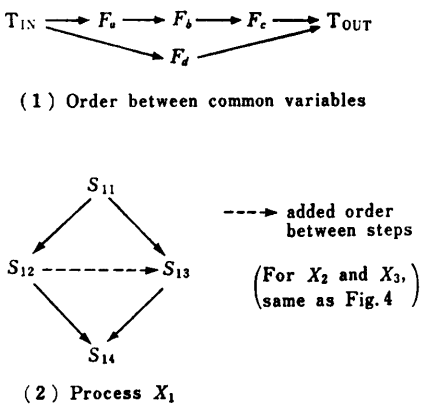

図 5 例題に対する相似化
Fig. 5 Similarization for the example system

ステップ間の順序関係へ逆写像すると，プロセス X_1 において順序対 $\langle S_{12}, S_{13}\rangle$ が追加される（図5(2)）．これによって，プロセス X_1 と X_3 はともに，$F_a \rightarrow F_c$ の順にアクセスを行うことになる．

以上の手続きは必ずしも一意の実行手順の選択を保証するわけではなく，例1～例3の場合でも，**図6**に示すようにプロセス X_3 に対しては選択の余地がある．臨界領域法ではこれ以上の選択は設計者にまかされている．ただし，次の第2段階の手続きは，それらのうちの任意の実行手順に対して適用可能である．

2.2.2 臨界領域の設定

2.1 節で述べた並行プロセスシステムの正当性の保証のために臨界領域（*CR*）[1] を用いる．ここで，*CR* は以下の性質を満足するものである．

（R1） *CR* は一度に一つのプロセスを到着順に受け付ける．

（R2） プロセスは *CR* 内に有限時間しかとどまらない．

（R3） 異なる *CR* 内での処理は並行に実行される．

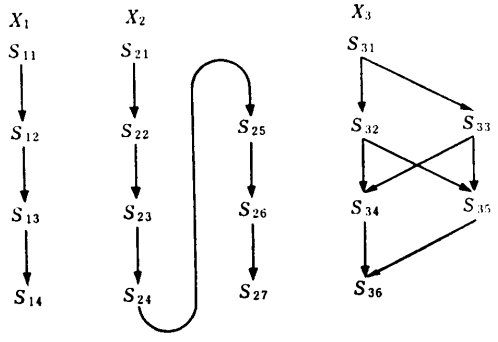

図 6 例題の実行手順
Fig. 6 Execution sequence for the example system.

また，後に述べるように，相互排斥やシリアライザビリティの保証のために *CR* のネストが必要となる場合があるが，直接的に *CR* のネストを用いるのではなく，次に述べるフロー制御臨界領域（*FCCR*）によって論理的にネストを実現する．

（F1） 各 *FCCR* は占有フラグ（セマフォ）をもつ．

（F2） プロセスは *FCCR* 内で入口ステップ（*P* 操作）を実行し，*FCCR* を出て必要な処理を行い，再度 *FCCR* に入って出口ステップ（*V* 操作）を行う．

（F3） あるプロセスがある *FCCR* の入口ステップを実行した後で，まだ出口ステップを実行していない場合（このとき占有フラグがオンになっている），他のプロセスが同一 *FCCR* の入口ステップを実行すると，そのプロセスは前のプロセスが出口ステップの実行を終了するまでキューに入って待つ．

以上の *CR* および *FCCR* を用いて次の三つのサブステップに従って設計をすすめる．

（C1） 相互排斥：各共有変数に対する相互排斥を実現する．

（C2） デッドロック防止：（C1）で *FCCR* が導入された場合，デッドロックのチェックを行い，対策を講じる．

（C3） シリアライザビリティ：複数の共有変数間の矛盾を防止し，各プロセスの処理環境のシリアライザビリティを保証する．

以下に各サブステップの詳細な手続きについて述べる．

相互排斥の保証は，各共有変数に *CR* を対応させて実現するが，それだけでは不十分である．あるプロセスが同一共有変数へ2度以上（不連続的に）アクセスする場合，途中で他のプロセスがその共有変数へアクセスするかもしれない．そこで，ある共有変数へ2度以上アクセスするプロセスが存在する場合，その共有変数に対して，*CR* のほかに *FCCR* も設定する．そして，その共有変数へアクセスするすべてのプロセスに，アクセスの前後で対応する *FCCR* の入口/出口ステップを実行させる．

［例4］ 例1のシステムで次のような実行手順

$X_1 : S_{11} \rightarrow S_{12} \rightarrow S_{13} \rightarrow S_{14}$

$X_2 : S_{21} \rightarrow S_{22} \rightarrow S_{23} \rightarrow S_{24} \rightarrow S_{25} \rightarrow S_{26} \rightarrow S_{27}$

$X_3 : S_{31} \rightarrow S_{33} \rightarrow S_{32} \rightarrow S_{34} \rightarrow S_{35} \rightarrow S_{36}$

を前提として，相互排斥手続きを適用すると，プロセス X_3 の実行手順は，

$X_3: S_{31} \rightarrow f^3_{cd} \rightarrow S_{33} \rightarrow S_{32} \rightarrow S_{34} \rightarrow S_{35} \rightarrow g^3_{cd} \rightarrow S_{36}$

となる．ここでは共有変数 F_d の相互排斥のために *FCCR* を設定し，プロセス X_3 の実行手順に入口ステップ f^3_{cd} と出口ステップ g^3_{cd} を挿入している．なお，F_d はプロセス X_3 内でしかアクセスされないが，プロセスはリエントラントな形で実行され，自分自身との競合もおこりうるので，*FCCR* の設定が必要である．

FCCR を含めてすべての臨界領域は形式的にはネスト状にはならないが，*FCCR* の場合には入口ステップから出口ステップまでが論理的に臨界領域を構成している．この論理的臨界領域のネストが存在する場合，デッドロックが発生する可能性がある．

デッドロック防止手続きでは，複数の *FCCR* 間のネストの順序関係を調べ，プロセスによってそのネストの順序が異なる場合には，*FCCR* の入口ステップの位置を調整することによって順序の同一化を図る．このための手続きは以下のようになる．

(D1) あるプロセスが $FCCR_a$ の入口ステップ f_a を実行後，出口ステップ g_a を実行する前に別の入口ステップ f_b を実行する場合，$\langle FCCR_a, FCCR_b \rangle$ の順序対を定義する．

(D2) このようにして定義された *FCCR* 間の順序関係にサイクルが存在する場合，そのサイクル上の順序対のうち一つを除去する．

(D3) 除去された順序対 $\langle FCCR_x, FCCR_y \rangle$ に対して，入口ステップ f_y を入口ステップ f_x の直前に実行するように，各プロセスの実行手順を変更する．

[例5] 二つのプロセス *Y*, *Z* の実行手順が，

$Y: S_{y1} \rightarrow f^y_A \rightarrow S_{y2} \rightarrow f^y_B \rightarrow S \rightarrow g_{y3} g^y_B \rightarrow g^y_A$

$Z: S_{x1} \rightarrow f^x_B \rightarrow S_{x2} \rightarrow f^x_A \rightarrow S_{x3} \rightarrow g^x_A \rightarrow g^x_B$

となっている場合，両プロセスが同時に S_{y2}, S_{x2} を実行するとデッドロックとなる．そこでプロセス *Z* の実行手順を，

$Z: S_{x1} \rightarrow f^x_A \rightarrow f^x_B \rightarrow S_{x2} \rightarrow S_{x3} \rightarrow g^x_A \rightarrow g^x_B$

のように変更する．

いくつかの変数の内容が互いに関連しているとき，それらへアクセスするプロセスの処理環境のシリアライザビリティのためには，プロセス間の同期によって変数間の相互無矛盾性を保持する必要がある．最も単純な同期は，関連する変数へのアクセスをすべて一つの（論理的な）臨界領域内で行うことである．しかし，2.1 節の図1で示したように変数へのアクセスの順序が同一であれば，シリアライザビリティを保証できる．この後者のほうが前者と比較すると，並行処理によってシステムの効率が向上する．このような点を考慮して以下のような手続きを行う．

(M1) 対象となる変数へアクセスしている処理ステップを含む実行手順の最小部分を各プロセスから抽出する．

(M2) 各処理ステップと臨界領域との関係に基づいて，抽出された実行手順に対応する臨界領域間の経路（領域間経路）を得る．ただし，*FCCR* の論理的臨界領域に含まれるステップは，ネストの最も外側の *FCCR* に対応させる．

(M3) すべての領域間経路が互いに等しいかどうかチェックする．異なる場合には次の(M4)または(M5)に従って領域間経路が等しくなるようにする．

(M4) あるプロセスに余分な処理ステップを挿入してみる．このときその処理ステップがアクセスする共有変数は，そのプロセスにとって新たな共有変数でなければならない．

(M5) 上の手続きが成功しない場合，新たな *FCCR* を導入し，領域間経路の異なる部分に対応する実行手順の部分に対して，その前後に入口/出口ステップを挿入する．これによって異なる部分が新たな *FCCR* に吸収され，すべての領域間経路が一致する．

[例6] 例4の結果に対してシリアライザビリティのチェックを行う．ここでは二つの共有変数 F_a と F_c のみが相互に関連していると仮定すると，プロセス X_1 と X_2 が問題になる．この二つのプロセスの領域間経路は，

$X_1: F_a \rightarrow F_c$

$X_2: F_a \rightarrow F_b \rightarrow F_c$

となるので，共有変数 F_b へアクセスする処理ステップ S_{15} をプロセス X_1 の実行手順中に

$X_1: S_{11} \rightarrow S_{12} \rightarrow S_{15} \rightarrow S_{13} \rightarrow S_{14}$

のように挿入すればよい．

[例7] 二つのプロセス *Y*, *Z* の実行手順が，

$Y: S_{y1}(A) \rightarrow S_{y2}(B) \rightarrow S_{y3}(C)$

$Z: S_{x1}(A) \rightarrow S_{x2}(B) \rightarrow S_{x3}(C)$

(A, B, C は共有変数)

となっている場合には，*FCCR* を導入し，

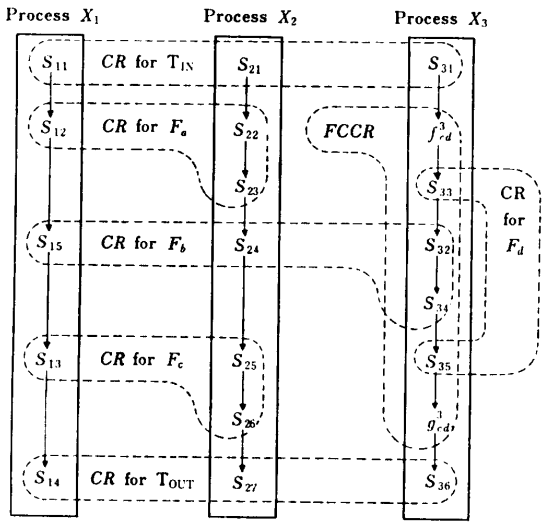

図 7 例題の設計結果
Fig. 7 Design result for the example system.

Y: $S_{y1}\rightarrow f^Y\rightarrow S_{y2}\rightarrow S_{y3}\rightarrow g^Y$

Z: $S_{z1}\rightarrow f^Z\rightarrow S_{z2}\rightarrow S_{z3}\rightarrow g^Z$

とすると，領域間経路はともに，

$Y, Z: A\rightarrow FCCR$

となって一致する．

以上の手続きによって各プロセスの実行手順が決定される．この手順に従うことによって，デッドロックの生じない正しい並行処理を実現できる．

［例 8］ 例 1 のシステムに対する最終の設計結果をまとめると，**図 7** のようになる．

3. 臨界領域法の定式化

臨界領域法は半順序集合上の操作として定式的に記述でき，それによって臨界領域法が矛盾を含まない正当なものであることが示される．

3.1 処理要求記述

並行プロセスシステムの処理要求記述は以下のように定義される．

(1.1.1) $PRD=\langle T, D, S, SD, TP\rangle$

(1.1.2) $T=\{t|$ プロセス$\}$

(1.1.3) $D=\{d|$ 共有変数$\}$

(1.1.4) $S=\cup S_t=\cup\{s_i|$ 処理ステップ$\}$ $(t\in T)$

(1.1.5) $SD: S\rightarrow D$ ($SD(s_i)$ は処理ステップ s_i でアクセスされる共有変数)

(1.1.6) $TP=\{P_t|$ プロセス t の処理要求$\}$ $(t\in T)$

(1.1.7) $P_t=\langle S_t, R_t\rangle$

(1.1.8) $R_t=\{\langle s_i, s_j\rangle|s_i, s_j\in S_t\}$ (S_t 上の半順序)

3.2 局 所 化

集合 S_t を以下の条件をみたすように部分集合へ分割する．

(2.1.1) $S_t=\cup LS_{ti}, LS_{ti}\neq LS_{tj}$ ならば $LS_{ti}\cap LS_{tj}=\phi$.

(2.1.2) $M_{td}=\{s_i|SD(s_i)=d, s_i\in S_t\}$ とすると，$s_i\in LS_{tj}$ かつ $s_i\in M_{td}$ なる s_i が存在するならば，任意の $s_n\in M_{td}$ は LS_{tj} に含まれる．

(2.1.3) $LS_{tk}(k=i1, i2, \cdots, in)$ 中にそれぞれ s_{atk}, s_{btk} が存在し，かつ順序対 $\langle s_{bti1}, s_{ati2}\rangle, \langle s_{bti2}, s_{ati3}\rangle, \cdots, \langle s_{btin}, s_{ati1}\rangle$ が R_t 中に含まれるならば，$LS_{ti1}=LS_{ti2}=\cdots=LS_{tin}$ となる．

集合 $\{LS_{ti}\}$ 上で関係 LR_t を次のように定義する．

(2.2.1) $R22_t=\{\langle LS_{ti}, LS_{tj}\rangle|{}^\exists\langle s_l, s_m\rangle\in R_t, s_l\in LS_{ti}, s_m\in LS_{tj}\}$

(2.2.2) $LR_t=R22_t$ の推移的閉包

［補助定理 2.3］ 関係 X がアサイクリックならば，X の推移的閉包 X' は非対称である．

(証明) X' 中に $\langle a, b\rangle$ が属するならば，$\langle a, x_1\rangle, \langle x_1, x_2\rangle, \cdots, \langle x_n, b\rangle$ となるような系列が X 中に存在しなければならない．したがって X' 中に $\langle b, a\rangle$ が存在するならば X 中にサイクルが存在することになり，前提条件に矛盾する．☒

［定理 2.4］ LR_t は $\{LS_{ti}\}$ 上の半順序である．

(証明) 反射律と推移律は定義より明らか．非対称律をみたすことは $R22_t$ がアサイクリックであることを示せばよい．これは $\{LS_{ti}\}$ の定義 (2.1.3) から成立する．☒

関係 LR_t を用いて関係 R_t を変形し，新しい関係 R_{tloc} を生成する．

(2.5.1) $R25_t=R_t\cup\{\langle s_l, s_m\rangle|s_l\in LS_{ti}, s_m\in LS_{tj}, LS_{ti}\neq LS_{tj}, \langle LS_{ti}, LS_{tj}\rangle\in LR_t\}$

(2.5.2) $R_{tloc}=R25_t$ の推移的閉包

［定理 2.6］ R_{tloc} は S_t 上の半順序である．

(証明) 反射率と推移律は明らか．非対称律に関しては $R25_t$ がアサイクリックであることを示せばよい．まず $R25_t$ の第 2 項が単独でサイクリックであることは，LR_t が半順序であることに矛盾する．そこで第 1 項と第 2 項の結合でサイクルができると仮定する．このとき R_t 中のみに属する $\langle a, b\rangle$ がサイクル上に存在し，$\{LS_{ti}\}$ の定義により，この二つの要素 a, b は同一の LS_{ti} に属する．したがって $R25_t$ 上にサイクルが存在するならば，その縮退サイクルが

$\{LS_{ti}\}$ 上の LR_t に存在することになり，定理 2.4 に矛盾する．

[系 2.7] R_{tloc} は R_t と矛盾する順序対を含まない．

(証明) R_{tloc} の定義および定理 2.6 より明らか．☒

3.3 相似化

共有変数の集合D上で次の関係 DR を定義する．

(3.1.1) $D31=\{\langle d_i, d_j\rangle \mid d_i, d_j\in D, \langle s_l, s_m\rangle \in R_t, s_l\in M_{td_i}, s_m\in M_{td_j}\}$

(3.1.2) $D32=D31$ の推移的閉包

(3.1.3) $DR=\{\langle d_i, d_j\rangle \mid d_i=d_j$ または $(\langle d_i, d_j\rangle \in D32, \langle d_j, d_i\rangle \notin D32)\}$

[定理 3.2] DR は集合D上の半順序である．

(証明) 反射律と非対称律は定義に含まれている．推移律の成立は以下のとおりである．まず，$\langle d_i, d_j\rangle$, $\langle d_j, d_k\rangle \in DR$ とすると，$\langle d_i, d_j\rangle \in D32$ かつ $\langle d_j, d_k\rangle \in D32$ より，$\langle d_i, d_k\rangle \in D32$ となる．一方，$\langle d_j, d_i\rangle \in D32$ と仮定すると．$\langle d_k, d_j\rangle \in D32$ となるが，これは $\langle d_j, d_k\rangle \in DR$ に矛盾する．ゆえに，$\langle d_i, d_k\rangle \in D32$ かつ，$\langle d_k, d_i\rangle \notin D32$，したがって $\langle d_i, d_k\rangle \in DR$ が成立する．☒

DR を用いて各プロセス t の半順序 R_t からそれぞれ新たな半順序 R_{tsim} を以下のように構成する．

(3.3.1) $R33_t=R_t\cup\{\langle s_i, s_j\rangle \mid SD(s_i)=d_l, SD(s_j)=d_m, d_l\neq d_m$ かつ $\langle d_l, d_m\rangle \in DR\}$

(3.3.2) $R_{tsim}=R33_t$ の推移的閉包

[定理 3.4] R_{tsim} は S_t 上の半順序である．

(証明) 定理 2.6 の証明と同様．☒

[系 3.5] R_{tsim} は R_t と矛盾する順序対を含まない．

3.4 臨界領域の設定

プロセス t の実行順序は次のように定義される．

(4.1.1) $Q_t=\{\langle s_i, s_j\rangle \mid s_i, s_j\in S_t\}$
ただし，$\forall s_i, \forall s_j, \langle s_i, s_j\rangle \in Q_t$
または $\langle s_i, s_j\rangle \in Q_t$

また，共有変数dに対する CR (臨界領域) M_d を，

(4.2.1) $M_d=\cup M_{td}=\{s \mid SD(s)=d\}$

とし，$FCCR$ (フロー制御臨界領域) M_c を，

(4.3.1) $M_c=F_c\cup G_c$
$=\{f_c \mid$ 入口ステップ$\}\cup\{g_c \mid$ 出口ステップ$\}$

とする．そして，全臨界領域の集合を CS とする．すなわち，

(4.4.1) $CS=\{CR\}\cup\{FCCR\}$

処理ステップの集合 S_t の拡大集合 AS_t を次のように定義する．

(4.5.1) $AS_t=S_t\cup\{$プロセス t の $FCCR$ 入口/出口ステップ$\}$

また，AS_t 上の実行手順を AQ_t とする．

(4.5.2) $AQ_t=\{\langle s_i, s_j\rangle \mid s_i, s_j\in AS_i\}$

3.5 相互排斥

相互排斥要求 ER を，

(5.1.1) $ER=\{$相互排斥の必要な共有変数$\}$ $(ER\subseteq D)$

とすると，共有変数 $d(\in ER)$ に対する相互排斥条件 $EC(d)$ は次のように定義される．

(5.2.1) $EC(d)=\mathrm{NOT}(\exists t\in T, \forall s_i, s_j\in M_{td}, \exists s_k \notin M_{td}, (\langle s_i, s_k\rangle \in AQ_t)\wedge(\langle s_k, s_j\rangle \in AQ_t))$
$\mathrm{OR}(\exists c\in CS, \forall t\subset T, \forall s\in M_{td}, (\langle f_c, s\rangle \in AQ_t)\wedge(\langle s, g_c\rangle \in AQ_t))$

[定理 5.3] 任意の d に対して $EC(d)$ を満足するような AS_t, AQ_t がすべてのプロセス t に対して存在する．

(証明) ある一つの $FCCR_c$ を設定し，それに対する入口ステップ f_c と出口ステップ g_c を各プロセスの実行手順 AQ_t の前後に付加することにより，$EC(d)$ の右辺第2項が必ず真になる．☒

3.6 デッドロック防止

$FCCR$ の集合 FM 上の関係 FR を以下のように定義する．

(6.1.1) $FM=\{FCCR\}$

(6.1.2) $F61=\{\langle M_a, M_b\rangle \mid M_a, M_b\in FM,$ ただし，$\exists t\in T, f_a, g_a, f_b, g_b\in AS_t$ かつ $(\langle f_a, f_b\rangle \in AQ_t)\wedge(\langle f_a, g_a\rangle \in AQ_t)\}$

(6.1.3) $FR=F61$ の推移的閉包

このときデッドロック防止条件 DC は以下のとおりである．

(6.2.1) $DC=(\forall M_a, \forall M_b(M_a\neq M_b)\in FM, \langle M_a, M_b\rangle \in FR\rightarrow\langle M_b, M_a\rangle \notin FR)$

一般に FR が上記のデッドロック防止条件 DC をみたすとは限らない．そこで，以下の制約に従って FD を構成する．

(6.3.1) $\langle M_a, M_b\rangle \in FR\wedge\langle M_b, M_a\rangle \notin FR \rightarrow\langle M_a, M_b\rangle \in FD$

(6.3.2) $\langle M_a, M_b\rangle \in FR\wedge\langle M_b, M_a\rangle \in FR \rightarrow\langle M_a, M_b\rangle \in FD\vee\langle M_b, M_a\rangle \in FD$

(6.3.3) $\langle M_a, M_b\rangle \in FD\rightarrow\langle M_b, M_a\rangle \notin FD$

(6.3.4) $\langle M_a, M_a\rangle \in FD$

(6.3.5) $\langle M_a, M_b\rangle \in FD \wedge \langle M_b, M_c\rangle \in FD \rightarrow \langle M_a, M_c\rangle \in FD$

注) (6.3.3)～(6.3.5) は FD が半順序となるための制約条件である.

FD を用いて新しい半順序 AQD_t を以下の手順で構成する.

(6.4.1) $NG_t = \{\langle s, g_a\rangle | {}^{\exists}M_a \in FM, g_a \in AS_t, \langle s, g_a\rangle \in AQ_t\}$

(6.4.2) $NF_t = \{\langle f_a, f_b\rangle | \langle f_a, f_b\rangle \in AQ_t \wedge (\langle g_a, f_b\rangle \in AQ_t \vee \langle M_b, M_a\rangle \notin FD)\}$

(6.4.3) $RF_t = \{\langle f_b, f_a\rangle | \langle f_a, f_b\rangle \in AQ_t \wedge \langle f_b, g_a\rangle \in AQ_t \wedge \langle M_b, M_a\rangle \in FD\}$

(6.4.4) $NB_t = \{\langle s, f_a\rangle | \langle s, f_a\rangle \in AQ_t \wedge ({}^{\forall}f_b, \langle f_a, f_b\rangle \notin RF_t)\}$

(6.4.5) $RB_t = \{\langle s, f_a\rangle | \langle s, f_a\rangle \in AQ_t \wedge ({}^{\forall}f_b$ such that $\langle f_a, f_b\rangle \in RF_t, \langle s, f_b\rangle \in AQ_t)\}$

(6.4.6) $NA_t = \{\langle f_a, s\rangle | \langle f_a, s\rangle \in AQ_t\}$

(6.4.7) $RA_t = \{\langle f_a, s\rangle | \langle s, f_a\rangle \in AQ_t \wedge ({}^{\exists}f_b, \langle f_b, s\rangle \in AQ_t, \langle f_a, f_b\rangle \in RF_t)\}$

(6.4.8) $AQD_t = Q_t \cup NG_t \cup NF_t \cup RF_t \cup NB_t \cup RB_t \cup NA_t \cup RA_t$

注) (6.4.1)～(6.4.7) の右辺において, s は入口ステップ (f_a, f_b など) 以外の任意のステップを示す.

[定理 6.5] AQD_t は AS_t 上の全順序である.

(証明) (6.4.8) の右辺の要素のうちで, $NF_t \cup RF_t$ は入口ステップの集合 $\{f_a\}$ 上の関係となっている. これが $\{f_a\}$ 上の全順序となることは, FR および FD の定義から示される. また, $NB_t \cup RB_t \cup RA_t$ は元の AQ_t において $\langle s, f_a\rangle$ の形であった順序対に対して, $NF_t \cup RF_t$ と矛盾するものを $\langle f_a, s\rangle$ と交換している. Q_t, NG_t, NA_t, は AQ_t にそのまま対応している. ゆえに AQD_t は AS_t 上の全順序となる. ☒

[定理 6.6] $\{AQD_t | {}^{\forall}t \in T\}$ は DC をみたす.

(証明) AQD_t は FD に矛盾しないように構成されている ((6.4.2) および (6.4.3)). すなわち, AQD_t に対して FR を求めると FD に一致し, FD は半順序であるので DC をみたす. ☒

[定理 6.7] $\{AQ_t | {}^{\forall}t \in T\}$ が ER 中の d に対して $EC(d)$ をみたすならば, $\{AQD_t | {}^{\forall}t \in T\}$ も $EC(d)$ をみたす.

(証明) AQD_t は AQ_t に対して入口ステップ (f_a など) のみを移動したものになっている. したがって $EC(d)$ の第1項は不変である. 第2項についても, 入口ステップがより前方に移動するのみなので, $\langle f_a, s\rangle \in AQ_t, \langle s, g_a\rangle \in AQ_t$ をみたす s は, AQD_t においても同じ条件をみたす. ゆえに, AQD_t に対しても $EC(d)$ が真になる. ☒

以上の定理 6.5～6.7 から, AQD_t を新しい AQ_t として以後用いることができる.

3.7 シリアライザビリティ

相互に内容の関連する共有変数の集合を MR とする.

(7.1.1) $MR = \{$相互に内容の関連する共有変数$\}$ $(MR \subseteq D)$

MR に対する関連処理要求は次のように定義される.

(7.2.1) $MT = \{t | t \in T, {}^{\exists}s \in S_t$ such that $SD(s) = d \in MR\}$

(7.2.2) $MS_t = \{s \in M_{td}$ for $d \in MR\} \cup \{s | s \in AS_t, {}^{\exists}M_a \in FM, BT(M_a, s) \wedge BT(M_a, s_d)$ for $s_d \in AS_t, SD(s_d) = d \in MR\}$

(7.2.3) $BT(M_a, s) = (\langle f_a, s\rangle \in AS_t) \wedge (\langle s, g_a\rangle \in AS_t)$

(7.2.4) $MQ_t = \{\langle s_i, s_j\rangle | s_i, s_j \in MS_t$ かつ $\langle s_i, s_j\rangle \in AQ_t\}$

次に, 処理ステップから臨界領域への写像 $SM_t(s)$ を定義する.

(7.3.1) $SM_t(s) = M_d$ (FM 中の任意の M_a に対して, $\langle s, f_a\rangle \in MQ_t$ または $\langle g_a, s\rangle \in MQ_t$)

(7.3.2) $= M_a$ ($M_a \in FM$, $\langle f_a, s\rangle \in MQ_t, \langle s, g_a\rangle \in MQ_t$, かつ FM 中の任意の $M_b (\neq M_a)$ に対して, $\langle s, f_b\rangle \in MQ_t$ または $\langle g_b, s\rangle \in MQ_t$)

$SM_t(s)$ を用いてプロセス t の領域間経路 NP_t は次のように定義される.

(7.4.1) $NP_t = \langle NS_t, NQ_t\rangle$

(7.4.2) $NS_t = \{M_i | {}^{\exists}s \in MQ_t, SM_t(s) = M_i\}$

(7.4.3) $NQ_t = \{\langle M_i, M_j\rangle | {}^{\exists}\langle s_k, s_l\rangle \in MQ_t, SM_t(s_k) = M_i, SM_t(s_l) = M_j\}$

[定理 7.5] $EC(d)$ が真ならば, NQ_t は NS_t 上の全順序である.

(証明) 定義より, 任意の M_i, M_j に対して s_k, s_l が存在する. MQ_t が全順序であるので, $\langle s_k, s_l\rangle$ ま

たは $\langle s_1, s_k \rangle$ が存在し，$\langle M_i, M_j \rangle$ または $\langle M_j, M_i \rangle$ が存在することになる．これらが全順序の条件をみたすことは，$EC(d)$ が真であること，および $SM_i(s)$ の定義によって導かれる．□

シリアライザビリティを保証するための条件 MC (MR) は次のように定義される．

(7.6.1) $MC(MR)=(NQ_{ii}=NQ_{ij}, {}^{\forall}t_i, {}^{\forall}t_j \in MT)$

[定理 7.7] $FCCR$ の挿入は任意の共有変数 d に対して相互排斥条件 $EC(d)$ をこわさない．

(証明) $EC(d)$ の第1項は無関係であり，第2項は存在条件なので，挿入によって条件の成立が影響をうけることはない．□

[定理 7.8] $FCCR$ の挿入は，それが NP_i を変化させる限り，デッドロック防止条件 DC に影響しない．

(証明) NP_i を変化させるためには，挿入される入口ステップ f_x は他のすべての f_a, g_a に対して，

$\langle f_x, f_a \rangle \in AQ_i$ または $\langle g_a, f_x \rangle \in AQ_i$

となっていなければならない．したがって，(6.1.2) より，FR 中に $\langle M_x, M_a \rangle$ は存在しうるが，$\langle M_a, M_x \rangle$ は存在しえない．ゆえに DC は真になる．□

4. おわりに

臨界領域法は，各プロセスの共有変数へのアクセスに着目して，それに対して適切な臨界領域を設定することを中心とした，並行プロセスシステムの設計手法である．臨界領域法の特徴は，(1)出発点となる処理要求記述が正しい限り，設計結果は並行処理に関する機能的正当性が保証される，(2)性能面をあまり問題にしない場合には，機械的に設計手続きをすすめることができる，という点にある．これらの特徴は，3章で述べたように，臨界領域法が半順序集合上の数学的操作として定式化されていることに基づいている．

謝辞 本研究に関してご討論いただいた図書館情報大学田畑孝一教授に感謝します．

参考文献

1) Hansen, P. B. : *Operating System Principles*, p. 55, Prentice-Hall, Englewood Cliffs, N. J. (1973).
2) Parnas, D. L. : A Technique for Software Module Specification with Examples, *Comm. ACM*, Vol. 15, No. 5, pp. 330-336 (1972).
3) Myers, G. J. : *Composite/Structured Design*, p. 1, van Nostrand, New York (1978).
4) Liskov, B. H. et al. : Specification Techniques for Data Abstraction, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 1, pp. 7-19 (1975).
5) Hansen, P. B. : The Programming Language Concurrent Pascal, *IEEE Trans. Softw. Eng.*, Vol. SE-1, No. 2, pp. 199-207 (1975).
6) Hirota, T. et al. : Computer-Aided Design of Software Module : Validity of Concurrent Processing on On-Line file, 3rd USA-JAPAN Computer Conference (1978).
7) Hirota, T. et al. : Automated Design of Concurrent Software Modules for On-Line File Processing, COMPSAC 80 (1980).
8) Coffman, E.G. Jr. and Denning, P. J. : Operating Systems Theory, p. 44, Prentice-Hall, Englewood Cliffs, N. J. (1973).
9) Bernstein, P. A. et al. : Formal Aspects of Serializability in Database Concurrency Control, *IEEE Trans. Softw.* Eng., Vol. SE-5, No. 3, pp. 203-216 (1979).
10) Bernstein, P. A. et al. : The Concurrency Control Mechanism of SDD-1 : A System for Distributed Databases (The Fully Redundant Case), *IEEE Softw. Eng.*, Vol. SE-4, No. 3, pp. 154-168 (1978).

(昭和57年11月22日受付)
(昭和58年3月11日採録)