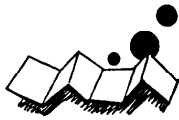


解説

データベースの同時処理制御における
直列可能性の理論†

室 章 治 郎††

1. ま え が き

データベースシステムにおける一つのまとまった処理単位（たとえば、利用者から入力される一つのプログラム）をトランザクションと呼ぶ。複数のトランザクションを処理しなければならないような状況下では、中央処理装置の遊び時間をできるだけ少なくし、処理効率を向上させ、各トランザクション終了までの待ち時間を平均化するために、それら複数のトランザクションを同時に並行処理するのが一般的である。ところが、むやみに同時処理を行おうとすると、データベース内の各データ間で満たされなければならない論理的な首尾一貫性 (consistency) が犯される可能性がでてくる。

たとえば、飛行機の座席予約システムのデータベースでは、予約された座席の数と残っている座席の数の和は、当然、常に一定でなければならない。予約された座席数 d の更新を行う次のプログラムを考えてみよう。

```
Read  $d$ ;
 $d := d + 1$ ;
Write  $d$ ;
```

今、AさんとBさんが同じ頃に別々の旅行社の窓口に行き、同じ飛行機の座席を予約しようとしたために、上のプログラムをほとんど同時に実行しようとする二つのトランザクション T_A , T_B があったとしよう。そのとき、 T_A と T_B のそれぞれ三つのステップを表-1 のように入れ違い (interleaved) に実行するスケジュール (入力系列) が形成される可能性がある。表-1 では、時間は右方へ向って進行し、 d の初期値は5であったとする。 T_A と T_B 両方とも、 d を1だけ増加させるプログラムだから、正しい結果は7であるべきであるが、 d の最終値は6となっている。

このような事態を首尾一貫性が犯されたという。この例では、 T_A , T_B のどちらか一方が、他方の実行された後の d の値を読めば問題が起きないのに (T_A が先に実行される場合を表-2 に示した)、両方とも T_A , T_B の実行前の d の値を読んでいることがデータ間の首尾一貫性を犯す原因となっている。

同時処理制御 (concurrency control) は、データベース管理システムにおける基本的な問題としてさまざまな立場から研究されているが、同時処理の妥当性の判定には、直列可能性 (serializability) の概念¹⁾が主として用いられている。つまり、トランザクションを表-2 のように一つずつ順次処理した場合と同じ結果になるようなスケジュールを直列可能であるといい、首尾一貫性を保証するための条件となっている。データベース管理システムでは、いかに直列可能なスケジュールを実現するかが一つの大きな課題であり、本稿ではこの問題について掘り下げて考えてみることにする。

今まで多くの文献で直列可能性について論じられてきたにもかかわらず、“誰”あるいは“何”に対して直列可能なスケジュールなのかということに関して曖昧なところがあった。そこで、直列可能な対象を明確にするうえで、一般的に考えられる四つの直列可能性の定義を紹介し、それらの定義のもとで直列可能なスケジュールのクラスに関して考察する。この議論は主に、Brzozowski と筆者による最近の研究^{8), 9)} をもとにしている。次に、直列可能性判定アルゴリズムについて論じる。そこでは最近、茨木、亀田、箕浦¹⁶⁾により直列可能性判定のための統一的な道具として提案された、トランザクション入力・出力グラフ (TIO グラフ) 上での Disjoint Interval Topological Sort (DITS) という概念を中心に説明する。また、直列可能性判定アルゴリズムに関する解説書などにあまり紹介されていない判定方法として、ある種の石置きゲーム (pebble game) を用いたものがある^{28), 36), 38)}。読者の中には石置きゲームに興味ある方も多いことと思

† On Serializability Theory in Database Concurrency Control by Shojiro MURO (Department of Applied Mathematics and Physics, Faculty of Engineering, Kyoto University).

†† 京都市工学部数理工学教室

表-1 首尾一貫性が保たれない例

T_A	Read d	$d := d+1$		Write d	
T_B	Read d	$d := d+1$		Write d	
データベースの d の値	5	5	5	5	6
T_A のワークスペースの d の値	5	5	6	6	6
T_B のワークスペースの d の値	—	5	5	6	6

表-2 首尾一貫性が保たれる例

T_A	Read d $d := d+1$ Write d
T_B	Read d $d := d+1$ Write d

い、その方法についても簡単な説明を加えた。最後に、現実性も含めて最近多くの論文で扱われている多版 (multiversion) モデルでの直列可能性 (たとえば、文献 1), 2), 6), 17), 26), 29), 30), 32), 33), 40), 41)) について概説する。

本稿では集中型データベースを想定して話を進めるが、分散型データベース (亀田²⁰⁾は、分散型データベースにおける諸問題を分かりやすく解説している。) においても、分散設置されたデータ間の首尾一貫性に注目すれば、本稿で論ずる直列可能性の理論はそのまま拡張できる。(分散型データベースの同時処理制御については Bernstein, Goodman⁵⁾ に詳しく、論文として 4), 21), 27), 34), 42), 43) など興味深い。) また、本稿ではロッキング機構 (たとえば、文献 13), 15), 24), 39), 44), 45)) などを用いて、いかに直列可能なスケジュールを生成するかについては (上林¹⁹⁾の解説文はこの話題を含んでいる) 力点を置かず、各直列可能性の定義のもとで直列可能なスケジュールの集合に注目し、その特性づけを中心に解説することにする。

2. システムモデル

本章では、システムモデルに関して以後の議論に必要な最小限の用語の説明を行う。

データベース管理システムには、トランザクション $T_1, T_2, \dots, T_n$ が入力されるものとする。各トランザクションが読んだり書いたりするデータの単位をデータ項目といい、データベースは、データ項目の有限集合

$$D = \{d_1, d_2, \dots, d_{|D|}\}$$

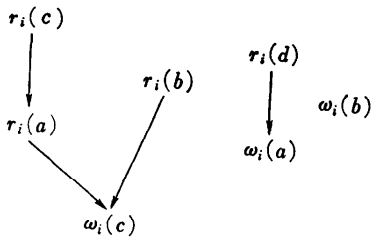
からなるものとする。ただし、 $|D|$ は D の位数を表す。

各データ項目 $d \in D$ に対して、同時処理を実行するうえで必要な操作として、たとえば、データの読み、書きの操作、ロック、ロック解除の操作が考えられ、さらにトランザクションを実行するには、(プログラム内での) いろいろな計算をする操作が必要である。それらの操作のうち、直列可能性に本質的に係わるものとして、データの読み込み操作と書き込み操作のみに注目し (以後、これらの操作を各々、**Read**, **Write** と記す)、各トランザクションはいくつかの **Read** と **Write** からなるものとする。ただし、データ項目 d へのトランザクション T_j の **Read** を $r_j(d)$, **Write** を $w_j(d)$ と記す。

トランザクションモデルとしては、一般モデル⁸⁾, 2 ステップモデル³¹⁾, 制約付 2 ステップモデル³¹⁾ の 3 種類を考えることにする*。一般モデルでは、 T_j を半順序 $\leq_j$ 上で定義されたいくつかの **Read** $r_j(d)$ と **Write** $w_j(d')$ の集合と捉える (ただし、同じ操作が 2 回以上現れることはない)。2 ステップモデルでは、 $r_j(X)(X \subset D)$ に $w_j(Y)(Y \subset D)$ が後続すると考える。ただし、 $A \subset B$ は A が B の部分集合であることを表す。ここで、 $r_j(X)(w_j(Y))$ は、各々 $X(Y)$ に属するすべてのデータ項目に対する同時の **Read** (**Write**) であり、個々の **Write** はすべての **Read** と $\leq_j$ の関係にあるものとする。 T_j にあるのが $r_j(X)$ と $w_j(Y)$ のどちらか一方のみでもよい。さらに、制約付 2 ステップモデルでは、 $X \subset Y$ という条件が課される。つまり、あるデータ項目 d の更新をするためには、その前に必ず d の値を読んでいなければならない⁴⁰⁾。

【例 1】一般モデルの例として、図-1 に示すトランザクション T_i が考えられる。ただし、図中の矢印は関係 $\leq_i$ を表す。また、2 ステップモデルとして、 $r_j(a, b, c, d, e)$, $w_j(a, c, e)$ からなるトランザクション T_j が考えられる。この T_j は制約付 2 ステップモデルでもある。各 $w_j(a)$, $w_j(c)$, $w_j(e)$ は、 $r_j(a)$, $r_j(b)$,

* トランザクションモデルについて Brzozowski⁴¹⁾ は詳しい議論をしている。

図1 一般モデルのトランザクション T_i

$r_i(c), r_i(d), r_i(e)$ すべてと $\leq_i$ の関係にある。□

$T_1, T_2, \dots, T_n$ に含まれる Read と Write は、あらかじめ各 T_i 内の関係 $\leq_i$ により指定された順序を除けば、任意の順序でデータベース管理システムに到着する。到着順にそれらの Read と Write を並べた実行系列をスケジュールという。たとえば、 $D = \{a, b\}$; $T_1: r_1(a) \leq_1 w_1(b)$; $T_2: r_2(a) \leq_2 w_2(a), r_2(a) \leq_2 w_2(b)$; $T_3: r_3(a) \leq_3 w_3(b)$ のもとで

$$e = r_1(a) r_2(a) w_2(a, b) r_3(a) w_1(b) w_3(b) \quad (1)$$

は、スケジュールの一例であり、 T_1, T_2, T_3 のすべての Read と Write を関係 $\leq_i (i=1, 2, 3)$ を満たした上でシャフルして得られる系列の一つとして与えられる。以後、与えられたスケジュールの実行に際しては、形式的な初期 Write $w_0(D)$ と最終 Read $r_f(D)$ が伴っているものとする。つまり、スケジュール e の実行前の各 $d \in D$ の値が $w_0(d)$ で表され、 e 実行後の d の値が $r_f(d)$ によって表されるものとする。

1章でも述べたように、直列なスケジュールとは、各トランザクションの Read または Write をトランザクションごとに順に並べた実行系列をいう。たとえば、上記の T_1, T_2, T_3 を添字の順に並べた直列なスケジュールとして次の s が考えられる。

$$s = r_1(a) w_1(b) r_2(a) w_2(a, b) r_3(a) w_3(b) \quad (2)$$

一般のデータベース管理システムでは、各データ項目 $d \in D$ のデータ値は（バックアップ用は別として）常に1個だけ保持されるので、あるスケジュール e の Read $r_i(d)$ は、 e において $r_i(d)$ より前に書かれた最新のデータ項目を読み込む。このようなモデルを単版（single-version）モデルという。それに対して、複数個（一般にはいくつでも）のコピーの存在を許せば、 $r_i(d)$ を実行する際にそれまでに書かれているコピーの中から適当なものを選んで読むことができ、6章で述べるように単版モデルと比べて同時処理の機能を高めることができる。このようなモデルを多版モデルという。

3. 直列可能性

“直列可能性”の定義は今まで論文などでいろいろな立場から定義され、数学的に統一されていなかった。そこで、それらの定義を見直し、Brzozowski⁸⁾, Brzozowski, Muro⁹⁾ をもとに4種類に類別^{*}して各々の定義のもつ意味について考察を加える。

あるスケジュール e を実行したあとで $d \in D$ に書かれている最終値 $r_f(d)$ の値を $d(e)$ と書くことにする。すべてのデータ項目の $d(e)$ の値に注目し、組 $D(e)$ を次のように定義する。

$$D(e) = (d_1(e), d_2(e), \dots, d_{|D|}(e))$$

スケジュール e の実行中に $r_i(d)$ によって読み込まれる $d \in D$ の値を $d(e)_i$ と記し、 T_i によって読み込まれるすべてのデータ項目の組 $(d_1(e)_i, \dots, d_{|D|}(e)_i)$ を $\tau_i(e)$ と書くことにする。さらに、 e の実行中にすべてのトランザクションによって読まれるデータ値の組 $\tau(e)$ を次のように与える。

$$\tau(e) = (\tau_1(e), \tau_2(e), \dots, \tau_n(e))$$

$w_i(d)$ の値は、 $r_i(d_j) \leq_i w_i(d)$ を満たすすべての値 $r_i(d_j)$ に依存している。つまり、これら $r_i(d_j)$ によって読み込まれた値をもとにして得られた結果の値が $w_i(d)$ によって書き込まれるものとする。この関係にある関数 $\Psi_{i,d}$ を導入して次のように書く。

$$w_i(d) = \Psi_{i,d}(r_i(d_1), \dots, r_i(d_n))$$

ただし、 $\{r_i(d_1), \dots, r_i(d_n)\} = \{r_i(d_j) \mid r_i(d_j) \leq_i w_i(d)\}$ 。

ここで、トランザクション $T_1, T_2, \dots, T_n$ から得られるすべてのスケジュールの集合 E 上での4種類の等価関係（equivalence relation）を定義する。

任意の $e, e' \in E$ と $i (=1, 2, \dots, n)$ に対して

$$(1) \quad e \sim_{\delta} e' \iff D(e) = D(e'),$$

$$(2) \quad e \sim_i e' \iff \tau_i(e) = \tau_i(e'),$$

$$(3) \quad e \sim_{\tau} e' \iff e \sim_i e' \text{ for all } i=1, 2, \dots, n,$$

$$(4) \quad e \sim_{\sigma} e' \iff e \sim_{\delta} e' \text{ and } e \sim_{\tau} e'.$$

(1), (3), (4) の等価関係を特に、 δ -等価、 τ -等価、 σ -等価と呼ぶ。これらの等価関係をもとに4種の直列可能性を定義しよう。あるスケジュール $e \in E$ は、

(a) (δ -直列可能である) $\iff$ (ある直列なスケジュール $s \in E$ が存在して $e \sim_{\delta} s$ が成立する),

(b) (τ -直列可能である) $\iff$ (E に属する直列なスケジュール $s_1, s_2, \dots, s_n$ が存在して $e \sim_i s_i (i=1,$

* 文献 8), 9) では5種類に類別しているが、本稿ではそのうちの4種類について考えることにする。

2, ..., n) が成立する),

(c) (τ -直列可能である) $\iff$ (ある直列なスケジュール $s \in E$ が存在して $e \sim_{\tau} s$ が成立する),

(d) (σ -直列可能である) $\iff$ (ある直列なスケジュール $s \in E$ が存在して $e \sim_{\sigma} s$ が成立する).

以上の δ, τ, σ の各直列可能性のもとで直列可能なスケジュールのクラスを各々, $\mathcal{D}, \mathcal{T}, \mathcal{S}$ と記し, 上記のスケジュールの集合 E も含めたすべてのスケジュールのクラスを $\mathcal{E}$, また, 直列なスケジュールのクラスを $\mathcal{S}$ と記す. 各クラスに属する例および各クラスの関係については, 次章を待たれたい.

δ -直列可能性は, 最終的にデータベースに書かれる値 $r_i(d) (d \in D)$ が首尾一貫性を満たしてさえいればよく, トランザクション (つまりは, システムの利用者) に提供されるデータについては関知しないというシステムサイドの考え方を反映した定義である.

逆に, τ あるいは τ -直列可能性では, トランザクションに提供されるデータの首尾一貫性を問題にし, データベースに書かれる値については問題にしないというシステムの利用者の立場をとる定義である. τ -直列可能性では, 各トランザクションに提供されるデータは直列なスケジュールを実行した結果であるものの, その直列なスケジュールが各トランザクションごとに異なってもよい. それに対し, τ -直列可能性では, すべてのトランザクションに一つの直列なスケジュールを実行したときと同じデータが提供される.

σ -直列可能性は, δ -直列可能性と τ -直列可能性を合わせた定義で, データベースに反映されるデータもすべてのトランザクションに提供されるデータも, ある一つの直列なスケジュール s を実行した結果であるという最も制約の強い定義である.

現実のデータベース上での各直列可能性の正当性に関して検討を加えてみよう. δ -直列可能という概念は Papadimitriou³¹⁾ によって導入されている. 直列可能性を考える際に, 関数 $\mathcal{P}_{i,d}$ を通じて, 各 $r_i(d) (d \in D)$ に影響を及ぼす Read, Write (このような Read, Write を活性 (live) であるという) を逆にたどっていき, これら活性操作のみを考慮し, 活性でない Read, Write については関知しないという立場である. その結果, あるシステムの利用者が論理的に首尾一貫していないデータ値を読み込む可能性があり, 問題が生じる場合が考えられる. その例として, 本稿の最初の飛行機の座席予約システムのデータベースにおいて, 予約座席数を確かめるだけのトランザクション

T_c があったとする. T_c は予約座席数が書いてあるデータ項目 d の読み込み操作 $rc(d)$ だけからなり, データ項目 d への書き込みをしないから活性ではないトランザクションである. したがって δ -直列可能性の定義からは, T_c が読み込む予約座席数が真の値でなくても一向に問題にされないというようなことが挙げられる. Papadimitriou が δ -直列可能性を用いた背景には, この定義のもとで直列可能性に関する本質的な諸特性を失うことなく, 数学的にきれいな議論ができることに重きを置いたのではないかと考えられる.

実際のデータベースシステムでは, 一旦システムが動き出してから次々とトランザクションが入力されてくるから, 上記の定義における最終 Read は便宜上のものとも考えられる. したがって, システムの利用者に首尾一貫したデータ値が提供されてさえいれば, データベースシステムとして十分機能を果しているとも考えられ, その点で τ あるいは τ -直列可能性の意義づけができる. τ -直列可能性については, Bernstein, Goodman⁶⁾ によっても論じられており, また, 最近, Yannakakis⁴⁵⁾, Rosenkrantz et al.³⁵⁾ も個々の利用者が対象とするデータベース (ビュー, view) の論理的な首尾一貫性を保証するクラスとして, その特性について論じている. 最終 Read の現実性を疑うことから τ -直列可能性の正当性を論じたが, $r_i(D)$ を文字通りの最終 Read と考えるのではなく, データベースの状態を調べるために, 同時にトランザクションが実行されていない任意の時点で挿入される Read とみなし, システムの故障なども考慮して, 保守的な立場からデータベースを首尾一貫した状態に保っておく重要さを考えると, δ -直列可能性を見直さなければならない.

一方, σ -直列可能性は Eswaran et al.¹³⁾, Stearns et al.⁴⁰⁾, Ibaraki et al.¹⁶⁾ などで用いられている. 制約の強さから現実的には同時処理能力は少々劣るにしても, 論理的には最も問題のない直列可能性の定義と考えられる.

以上のように直列可能性の定義については, いろいろな立場から正当性, 有用性を論じることができ, どれが最適であるかという判断は難しいように思われる.

4. 各直列可能性のクラスの同時処理能力

前章で定義した各直列可能性のもとでのスケジュー

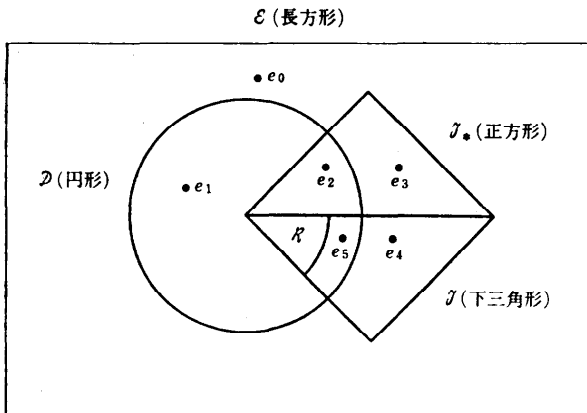


図-2 一般モデル, 2ステップモデルの各クラスの包含関係

ラの同時処理能力を、各直列可能なスケジュールのクラス $\mathcal{D}$, $\mathcal{G}_*$, $\mathcal{G}$, $\mathcal{R}$ に関して Brzozowski⁸⁾, Brzozowski, Muro⁹⁾ によって得られている包含関係を示すことにより比較したい。ただし、単版モデルのみを対象とし、一般モデル, 2ステップモデル, 制約付2ステップモデルの各々のトランザクションモデルについての結果を紹介しよう。多版モデルに関する結果は6章で紹介する。

まず、一般モデルと2ステップモデルの $\mathcal{D}$, $\mathcal{G}_*$, $\mathcal{G}$, $\mathcal{R}$ の包含関係については差異が生じないので、2ステップモデルに関する結果を図-2に示した。図中の各クラスに属するスケジュールの列を $e_0, e_1, \dots, e_5$ として示した。ただし、これらの例は次のように与えられる。

$$\begin{aligned} e_0 &= r_1(a)r_2(b)w_2(a)w_1(b)r_3(a,b) \\ e_1 &= e_0w_1(a,b) \\ e_2 &= r_1(a,b)r_2(a)w_2(a)r_3(a,b)w_1(b) \\ e_3 &= r_1(a)r_2(a)w_1(a)w_2(a) \\ e_4 &= r_2(a)w_1(a)w_2(a) \\ e_5 &= r_1(a)w_1(b)r_2(b)w_3(b,d)r_4(d)w_4(a,c,e)w_5(b,e) \\ &\quad r_6(e)w_6(a,c,d)w_2(c)w_7(a,b,d,e) \end{aligned}$$

まず、 $e_0 \notin \mathcal{D} \cup \mathcal{G}_*$ となることについて考えてみよう。 $r_1(a), r_2(b)$ は、 $w_0(a), w_0(b)$ によって書かれた値をそれぞれ読んでおり、Write $w_1(b), w_2(a)$ があることから、 T_1, T_2 それぞれにとっては、直列なスケジュール $T_1T_2T_3, T_2T_1T_3$ と τ -等価である。ところが、 $r_3(a,b)$ については、 $r_3(a), r_3(b)$ はそれぞれ T_1T_2, T_2T_1 と実行された結果の値を読むことから、 $e_0 \notin \mathcal{G}_*$ である。さらに、 $r_1(a,b)$ と $r_3(a,b)$ は同じ a, b の値を読むことより、 $e_0 \notin \mathcal{D}$ も同様に示せる。

また、 $e_1 \in \mathcal{D} \cap \mathcal{G}_*$ であることは次のように示せる。 e_0 に $w_1(a,b)$ をつけ加えることにより、 $r_1(a,b)$ によって読まれる a, b の値は $w_1(a,b)$ によって書かれたものとなり、 $w_1(a), w_1(b)$ の値はほかのトランザクションによって書かれた値に依存していないことより、 e_1 は、たとえば、直列なスケジュール $T_1T_2T_3T_4$ と δ -等価となる。しかし、依然、 $e_1 \notin \mathcal{G}$ であることは、 e_0 の場合と同じ理由で示せる。ほかの e_2, e_3, e_4 が図中の位置を占めることについては読者に考えて頂くことにして、次の興味深い結果について考えてみよう。

【定理1】⁹⁾ 2ステップモデルにおいて $\mathcal{D} \cap \mathcal{G} \supseteq \mathcal{R}^*$ が成立する。

σ -直列可能性は、 τ -直列可能性と δ -直列可能性を兼ね備えたものであるから、一見、 $\mathcal{D} \cap \mathcal{G} = \mathcal{R}$ と考えがちであるが、実は、 $\mathcal{D} \cap \mathcal{G}$ は $\mathcal{R}$ を真に含んでいることを e_5 によって示せる。

まず、 e_5 の各トランザクションを直列に並べた次のスケジュール s_5 を考えよう。

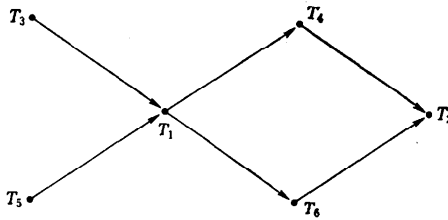
$$s_5 = T_3T_6T_1T_4T_6T_2T_7$$

T_7 は Write のみからなり（つまりほかのトランザクションとの依存関係がない）、 e_5, s_5 両方のスケジュールにおいて a, b, d, e の最終値を書き込む。残りの c の値について、両方のスケジュールにおいて、最終値はともに $w_2(c)$ によって書かれ、 $w_2(c)$ は $r_2(b)$ に依存し、 $r_2(b)$ は $w_1(b)$ の値を読み、 $w_1(b)$ は $r_1(a)$ に依存し、 $r_1(a)$ は $w_0(a)$ の値を読み込むという関係が成立している。したがって $e_5 \sim s_5$ 、つまり、 $e_5 \in \mathcal{D}$ が成立する。次にもう一つの直列なスケジュール t_5 を以下のように与える。

$$t_5 = T_1T_2T_3T_4T_6T_5T_7$$

e_5 と t_5 では、各トランザクションのデータの読み込みに関しては同一の関係をもっていることより、 $e_5 \sim t_5$ が成立し、 $e_5 \in \mathcal{G}$ であることは容易に分かる。最後に $e_5 \in \mathcal{R}$ を示そう。今、ある直列なスケジュール u_5 が存在して $e_5 \sim s_5u_5, e_5 \sim t_5u_5$ が成立すると仮定しよう。まず、 T_4, T_6, T_2 はいずれもデータ項目 c への Write をもち、しかも c の最終値は T_2 によって書き込まれていることより、 T_4 と T_6 は T_2 より前に実行されなければならない。次に、 T_4, T_6 はともに a への Write をもつが、 T_1 は $w_0(a)$ を読み込んでいることより、 T_1 は T_4, T_6 より前に実行されなけ

* $A \supseteq B$ は集合 A が集合 B を真に含むことを示す。

図-3 $e_6 \notin R$ を示す過程で得られる部分順序関係

ればならない。さらに、 T_2 は T_1 の書いた b の値を読み、 T_3 と T_5 は b の値を書き込んでいることより、 T_3 、 T_5 のいずれも T_1 の実行と T_2 の実行との間に入り込むことはできない。もし T_3 が T_2 の後に実行されると、 T_3 の書いた d の値を読んでいる T_4 が T_2 の後に実行されることになり、最初の結果に矛盾する。したがって、 T_3 は T_1 より前に実行されなければならない。同様に、 T_5 は T_1 より前に実行されなければならない。以上をまとめると 図-3 に示したような部分順序関係を得る。図-3 より全順序の直列なスケジュールを導くために、 $T_5 \rightarrow T_4 \rightarrow T_6$ または $T_6 \rightarrow T_4$ とする必要がある。前者のようにすると $r_6(e)$ が $w_6(e)$ の値を読むことになり矛盾し、後者のようにすると $r_4(d)$ が $w_6(d)$ の値を読むことになりやはり矛盾する。結果的には直列なスケジュール u_6 は存在することができなくなり、 $e_6 \notin R$ が示される。

以上のようにして、 δ -等価な直列なスケジュールと τ -等価な直列なスケジュールとして同一のものを取り得ないスケジュールが $\mathcal{D} \cap \mathcal{R}$ に属することが 2 ステップモデルで存在することが明らかになった。

図-2 には明示していないが、 $R \supseteq S$ が成立することを示すためにスケジュール

$$e_6 = r_1(a)r_2(b)w_1(a)w_2(b)$$

を考えよう。 e_6 は直列ではないが、 T_1 は a のみ、 T_2 は b のみに関係していることより、 σ -直列可能であることは明らかである (e_6 は制約付 2 ステップモデルの場合にも $R \supseteq S$ であることを示している)。

さて、トランザクションモデルを制約付 2 ステップモデルにすると、一般モデル、2 ステップモデルとは異なった包含関係が得られ、図-4 のようになる。ただし、図中の e'_6 は

$$e'_6 = r_1(a, b)r_2(a, b)w_1(a)w_1(b)r_2(a, b)$$

で与えられる。次の定理が成立することは興味深い。

【定理 2】⁹⁾ (a) 制約付 2 ステップモデルにおいて、 $\mathcal{G} = \mathcal{R}$ 、 $\mathcal{D} = \mathcal{G} * \cap \mathcal{D}$ が成立する。(b) Read だけからなるトランザクションがない制約付 2 ステップモデル

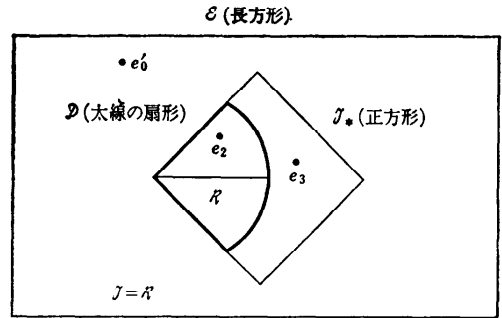


図-4 制約付 2 ステップモデルの各クラスの包含関係

ルでは $\mathcal{G} = \mathcal{D} = \mathcal{R}$ が成立する。□

以上、単版モデルについて各直列可能性の同時処理能力を、各定義のもとで直列可能となるスケジュールの包含関係を明らかにすることにより検討してみた。直列可能性についての諸性質について調べてみると、一般モデルに関して得られる結果は、2 ステップモデルでも同じ結果が示され、制約付 2 ステップモデルにすると異なった結果になることが往々にしてあるものであるが、以上の議論でも同様の傾向が示されたことは興味深い。直列可能性に関して最もよく参照される Papadimitriou の論文³¹⁾で 2 ステップモデルを用いて議論を展開しているのが、このような傾向を見出した上でのことであるとすれば、その洞察力には驚かざるを得ない。

5. 直列可能性の判定法

直列可能性の判定、つまりデータベース管理システムのスケジューラが、与えられたスケジュールが直列可能かどうかをいかに判定するかという問題は、その判定のアルゴリズムの複雑さの研究も含めて同時処理制御の中心的な話題の一つである。本質的には、スケジュール内に現れる Read、Write の各データ項目に関する実行順序の競合関係をなんらかの方法で表現し、直列なトランザクションの実行の場合と比較し、前章で論じた意味で等価であるかどうかを判定する方法の開発である。今までに直列可能性を判定する道具としてたくさんの方が、直列可能性のクラスの提案とともになされてきており、枚挙にいとまがないほどである。本稿ではそれらの内で、視覚的にも直列可能性という性質を把握しやすく、しかも今まで解説文などであまり扱われていない、Ibaraki et al.¹⁶⁾で導入された DITS による方法と、箕浦²⁸⁾、Sethi^{36), 38)}らによって論じられている石置きゲーム (pebble game)

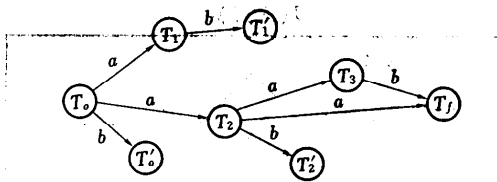
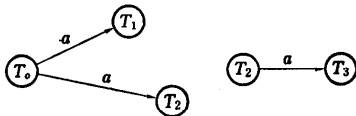
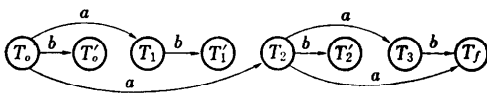
図-5 トランザクション入力・出力グラフ $TIO(e)$ 

図-6 図-5 中のインターバルの例

図-7 図-5 のグラフ $TIO(e)$ に対する DITS

を用いる方法の二つを紹介しよう。本章での議論は σ -直列可能性の場合に限って行うが、 δ -直列可能性、 τ -直列可能性の場合にも DITS の概念が拡張できることは木庭ほか²⁵⁾に見られる。

5.1 DITS と直列可能性

与えられたスケジュール e に対し、トランザクション入力・出力グラフ $TIO(e)$ を次のように定義する。 $TIO(e)$ は、 e に現れる各トランザクション T_i と $w_o(D)$ に対応する T_o 、 $r_f(D)$ に対応する T_f を節点とし、 $w_s(a)$ で書かれた a を $r_i(a)$ が読むとき、ラベル a をもつ有向枝 (T_s, T_i) を設ける。 $w_s(a)$ によって書かれた a がどの Read によっても読まれないとき、 $w_s(a)$ を無効 (useless) であるといい、 T_s にダミー節点 T'_s を付し、ラベル a をもつ有向枝 (T_s, T'_s) を書く。式(1)の e に対する $TIO(e)$ は図-5 のようになる。ある節点から発し、同じラベルをもつ枝の集合をインターバルという。有向閉路をもたない $TIO(e)$ に対し、 $TIO(e)$ の全節点を左から右に枝方向に矛盾しないように一列に並べるとき、同じラベルをもつ二つ以上のインターバルが重ならないければ、DITS であるという。図-5 の中でインターバルの例を図-6 に、図-5 に対する DITS を図-7 に示した。

以上の例で示した定義のもとで次の定理が成立する。

[定理 3]¹⁶⁾ $e \in \mathcal{R}$ の必要十分条件は、 $TIO(e)$ に DITS が存在することである。 $\square$

【例 2】 式(1)の e に対する 図-5 の $TIO(e)$ は有向閉路をもたず、しかも図-7 の DITS が存在する。したがって図-7 の DITS から導かれる式(2)の直列なスケジュール s と σ -等価であり、 $e \in \mathcal{R}$ が成立する。 $\square$

5.2 条件付直列可能性

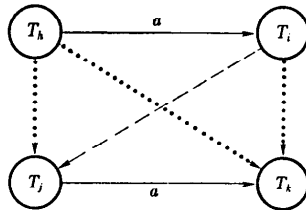
直列可能性の判定問題は 5.3 節で述べるように、一般モデル、2ステップモデルに対しては NP-完全¹⁴⁾であり、効率の良い判定アルゴリズムが存在するとは考えられない。そこでトランザクションモデルに制約を加える (たとえば制約付 2ステップモデル) という方向とは別に、直列可能性の定義にいくつかの付加条件を設けることにより、判定が多項式時間であるようなクラスの構成が試みられてきた。代表的なものとして、Papadimitriou³¹⁾ による DSR (Bernstein et al.³⁾ の CPSR と同一) がある。ただし、これら条件付直列可能性の議論が、定義されるクラスごとに異なる判定の道具を用いてなされてきたために、本質的には同じクラスであっても外見上異なったものと見られているというように、全体の見通しが悪いという問題があった。Ibaraki et al.¹⁶⁾ は、前節で導入した DITS という概念を同一の道具として一貫して用いることにより、今まで定義されてきた種々の条件付直列可能性のクラスの構造上の相違を明確にすることに成果を収めている。

DSR を始めとして条件付直列可能性の種々のクラスは、与えられたスケジュール内の同じデータ項目に関しての Read, Write の前後関係のある部分を保存するという付加条件によって定義されていることに注目し、まず、次のような付加条件が導入された¹⁶⁾。

σ -直列可能性の定義において、与えられたスケジュール e と σ -等価となる直列なスケジュール s に関して次の付加条件を考える。

WW-条件: あるデータ項目 a に対し、 e において $w_i(a)$ が $w_j(a)$ より前にあれば、 s においても T_i は T_j よりも前に実行される。

同様に、 $w_i(a)$ と $r_j(a)$ の対、 $r_i(a)$ と $w_j(a)$ の対、 $r_i(a)$ と $r_j(a)$ の対の場合を考えて、**WR-条件**、**RW-条件**、**RR-条件**が定義される。WW-条件の下で σ -直列可能なスケジュールのクラスを WW と記す。WR, RW, RR のクラスも同様に定義される。また、たとえば、クラス $WR+RW$ は、WR-条件と RW-条件の両方を満たしたうえで σ -直列可能なスケジュールのクラスを示す。他の組み合わせについても

図-8 TIO(e) に付す WW-条件を表す条件枝

同様に定義する。このようにして定義された条件付直列可能性の判定問題が以下に示すようにすべて DITS の概念で扱えることは、DITS の汎用性を裏付けるのに十分である。

条件付での σ -直列可能性を判定するために、付加条件に対応して TIO(e) に条件枝を付す必要がある。たとえば WW-条件の場合、 e において $w_i(a)$ が $w_j(a)$ より前にあれば、 T_i は T_j より前に実行されることを示す有向枝 (T_i, T_j) を加える。この有向枝はトランザクション T_i と T_j の実行の前後関係を示すもので、データ項目単位の関係を示すものではないのでラベルは付さない。これらの枝と DITS の条件からさらに枝を追加する必要がある。図-8 に示したように、ラベル $a \in D$ をもつ有向枝 (T_h, T_i), (T_j, T_k) が TIO グラフにある場合 (T_i と T_k はダミー節点であってもよい)、点線の有向枝 (T_h, T_j), (T_h, T_k), (T_i, T_k) のどれか一つでも TIO グラフに加われば、DITS が存在するためには明らかに T_i を T_j に先行させる必然性が生ずる。その状況を示すために破線の枝 (T_i, T_j) が付加される。この破線枝の追加は本質的には、Sethi^{[36], [37]} による排除規則 (exclusion rule) と同一のことである。得られたグラフを TIO_{WW}(e) と書く。他の条件付直列可能性のクラスについても同様のグラフを定めることができる。たとえば、RW においては、 e において $r_i(a)$ が $w_j(a)$ より前にあれば有向枝 (T_i, T_j) を加え、さらに先程の排除規則の枝も加えて TIO_{RW}(e) グラフが得られる。このようにして条件付直列可能性の各クラスに対して作られる TIO グラフに対して、次の定理が示される。

【定理 4】^[16] 各クラス $* = WW, WR, RW, \dots$ などに対し、 $e \in *$ であるための必要十分条件は、TIO_{*}(e) に DITS が存在することである。□

5.3 条件付直列可能性のクラスの包含関係

一般モデル、2ステップモデルのトランザクションモデルに対して、前節で導入した WW, WR+RW などのクラスの間の包含関係を 図-9 に示した。図中

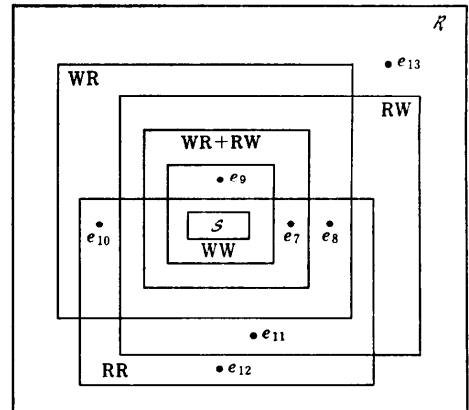


図-9 条件付直列可能性の各クラスの包含関係

のスケジュール例を 2 ステップモデルで Ibaraki et al.^[16] より引用して以下に与えるので、興味ある読者は確かめられるとよい。

$$\begin{aligned} e_7 &= r_1(a)r_2(a)w_2(a,b)r_3(a)w_1(b)w_3(b) \\ e_8 &= w_1(a)r_2(a)w_3(a)w_2(a)r_4(a)w_5(a) \\ e_9 &= r_2(a)r_1(a)w_2(a) \\ e_{10} &= r_3(a)w_1(a)r_2(a)w_3(a)w_2(a) \\ e_{11} &= r_2(b)r_1(a)w_2(a)w_1(a)r_3(a)w_4(a) \\ e_{12} &= e_{10}r_1(a,b)w_1(a,b)e_{11} \\ e_{13} &= e_{10}r_1(a,b)w_1(a,b)e_{11}r_2(a,b)w_2(a,b)e_9 \end{aligned}$$

図-9 で特に興味深いのは、

$WW = WW + WR + RW = WW \cap WR \cap RW$ の成立することである。Papadimitriou^[31] の DSR は、本来 $WW + WR + RW$ として定義されているが、この結果から WW に等しく、 $WR + RW$ という付加条件は冗長であることが判明した。

さらに制約付 2 ステップモデルにすると、図-10 のように包含関係が異なってくる。 $WW = WR = RW =$

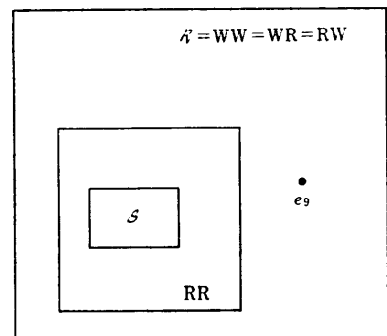


図-10 制約付 2 ステップモデルにおける条件付直列可能性の各クラスの包含関係

表-3 各クラスの所属判定アルゴリズムの計算の複雑度

計算の複雑度	σ -直列可能性のクラス
多項式時間	WW, WR+RW
NP-完全	$\mathcal{R}$, WR, RW, RR, WR+RR, RW+RR

$\mathcal{R}$ となってしまうが、 δ -直列可能性の場合にも WW に対応するクラス DSR に対して $DSR = \mathcal{D}$ となることが Papadimitriou³¹⁾ により示されている。

5.4 判定アルゴリズムの計算の複雑度

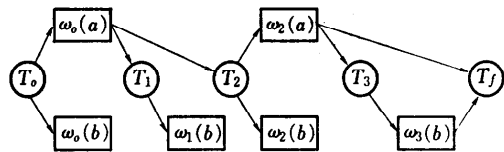
表-3 に σ -直列可能性の場合の各クラスの所属問題の計算の複雑さに関して、Ibaraki et al.¹⁶⁾ により得られている結果をまとめて示した。 δ -直列可能性の場合にも $\mathcal{D}$ の所属問題は NP-完全であり、WW に対応する DSR では多項式時間で判定できることが Papadimitriou¹⁶⁾ により示されている。表-3 より、多項式時間アルゴリズムをもつ最も大きなクラスが WR+RW であることが明らかになったが、WR+RW が WW を真に含んでいることより (図-9 参照)、直列可能性の定義の相違はあるにせよ、本質的には今のところ WR+RW が多項式時間で所属問題が判定できる最大のクラスと考えられる。

5.5 石置きゲームと直列可能性

読者の中には石置きゲームに興味を持っておられる方も多いと思われる。そこで、直列可能性の判定問題がある種の石置きゲームに置き換えられることを簡単に説明する。

まず、与えられたスケジュール e に対してゲームグラフ $PG(e)$ を次のように定義する。 $PG(e)$ は、各トランザクション T_i と各トランザクション T_i の Write $w_i(a)$ を節点とする。ただし、 $w_i(D)$ をもつ T_i 、および、 $r_j(D)$ をもつ T_j も $TIO(e)$ の場合と同様に節点の集合に加えるものとする。 T_i と Write $w_i(a)$ に対して有向枝 $(T_i, w_i(a))$ を付し、write 枝といい、 $w_i(a)$ を T_i の出力節点という。また、 T_i が Read $r_i(a)$ をもち、 $w_j(a)$ によって書かれた値を読み込むとき、有向枝 $(w_j(a), T_i)$ を付し、read 枝といい、 $w_j(a)$ を T_i の入力節点という。たとえば、(1)式の e に対してゲームグラフ $PG(e)$ を書くと図-11 のようになる。 $PG(e)$ は、グラフ $TIO(e)$ のラベル付き有向枝のかわりに各トランザクションの書いた値の節点を設け、その節点を通じて各データ項目の値の読み書きの情報を提供するものである。

このゲームグラフ $PG(e)$ 上で、各トランザクションがスケジュール e で与えられるのと同じデータの読

図-11 ゲームグラフ $PG(e)$

み込み、書き込みを行った上で、各トランザクションの実行が一瞬に (instantaneously) 実行されるような状況 (つまり、他のトランザクションの実行と混ざらない直列な実行) が作り出せるかどうかをテストするのが、直列可能石置きゲームテスト (serializability pebble game test, SPGT と略す) である。まず、直列可能石置きゲーム (SPG) のルールを記そう。

SPG ルール: 最初、 T_0 の出力節点のすべて $w_0(a)$ ($a \in D$) に a とラベル付けされた石を置く。各節点 T_i は、 T_i の入力節点すべてに石が置かれたら、 T_i の出力節点 $w_i(a)$ のすべてに a とラベル付けされた新しい石を置き、もし、 a とラベル付けされた古い石が $PG(e)$ 内にあったら、その古い石を取り除く、新しい石を置く操作と古い石を取り除く操作は同時に (indivisibly) 行われるものとする。□

SPG ルールのもとですべてのトランザクションが各1回石の受け渡しをし、しかも、最後に T_j の入力節点にすべての石が置かれるような状態になるとき、 $PG(e)$ 上で SPGT が完了したという。

[定理5]²⁸⁾ 与えられたスケジュール e が σ -直列可能であるための必要十分条件は、ゲームグラフ $PG(e)$ 上での SPGT が完了することである。□

e と σ -等価な直列なスケジュールは、石を受け渡していった順にトランザクションを並べたものとなる。ただし、トランザクション間でどのような順序で石を受け渡したら良いかということは難しい問題であり、SPGT の完了問題を考えたからといって直列可能性の問題がやさしくなるということではない^{36), 38)}。

図-11 上で、SPGT テストが完了することを確認してみられたい。また DITS の場合と同様に、グラフ $PG(e)$ に有向枝を付加することにより、WW などにおいて、与えられたスケジュールの所属判定にも拡張して用いることができる²⁸⁾。

6. 多版モデル

これまでの議論は、各データ項目のコピーは常に1個だけ存在し、与えられたスケジュール e 内の $r_i(a)$ は、常に e 内で $r_i(a)$ のすぐ前に現れる a への書き込

み命令によって書かれた値を読むという前提のもとで話を進めてきた。ところで、各データ項目 a の値として、それまでに a に書き込まれた古いコピー (版, version) の存在 (一般的には任意の数) を許せば、各読み込み操作 $r_i(a)$ を実行するに際して、それまでに書かれているコピーの中から適当なものを選んで読むことができる。その結果、単版方式のもとでは直列可能でないスケジュールも多版方式のもとでは直列可能となる可能性があり、同時処理の機能を高めることができる。

たとえば、簡単な例として次のようなスケジュールを考えよう。

$$e' = r_1(a)w_2(a)w_2(b)r_1(b)$$

単版方式のもとで明らかに、 $e' \sim_s T_1 T_2 (e' \sim_s T_2 T_1)$ でもある) であるが、 e' は τ, τ_*, σ のいずれの直列可能でもない。これは $r_1(a)$ が $w_2(a)$ の書いた値を、 $r_1(b)$ が $w_2(b)$ の書いた値を読んでいるからである。そこで、 $w_2(b)$ が書いた版ができて $w_2(b)$ の版を残しておき、 $r_1(b)$ に古い方の版を読ませれば、その実行結果は σ -直列可能 (つまり τ, τ_* -直列可能) となる。

与えられたスケジュール e 内の各 Read $r_i(d)$ に対して、 $r_i(d)$ に先行する一つの d への Write $w_j(d)$ に割り当てたうえでのスケジュールの実行結果が、3章で定義した意味で、 $*$ ($*$ = $\delta, \tau, \tau_*, \sigma$) -直列可能であるならば、 e は多版モデルにおいて $*$ -直列可能であるといい、各々のクラスを $\hat{\mathcal{D}}, \hat{\mathcal{D}}, \hat{\mathcal{D}}^*, \hat{\mathcal{R}}$ とそれぞれ記す。

多版モデルにおいて直列可能性を論じるとき、最終 Read $r_f(d) (d \in D)$ の解釈に関して二通り考えられる。最初の解釈は、 $r_f(d)$ も読み込み操作であるから、システム内の d の版のどれを読んでもよいという立場をとるものである。もう一方は、 $r_f(d)$ は架空の Read であり、一般のトランザクションの Read とは異なり版の選択はできず、 $r_f(d)$ を実行する時点でデータ項目 d の代表値、つまり、その値として最もふさわしい最新版の d の値が $r_f(d)$ に読まれるものとして与えられると解釈する立場である。後者では、スケジュールを実行する各時点においては、各データ項目 $d \in D$ は複数個の版をもち同時処理能力をあげることができるが、各時点での d の代表値は一つであると考えられる。たとえば、銀行に行って残高照会をしたとき、返答される値は残高の最新値であり、その時点までの残高が次々と出力されては困るというような状況

を反映している。

Bernstein, Goodman⁶⁾, Papadimitriou, Kanellakis³²⁾, Ibaraki, Kameda¹⁷⁾ などは前者の定義を用い、Brzozowski, Muro⁹⁾ は後者の考え方をを用いている。どちらの定義においても、与えられたスケジュール e 内のすべての Read $r_i(d)$ が $w_0(d)$ を読むようにすれば、 e は T_i を最初に行う任意の直列なスケジュール s_i と $\sim_i$ の意味で等価になることより、 $\hat{\mathcal{D}}^* = \mathcal{D}$ が成立する。前者の定義のもとで各トランザクションモデルにおける $\hat{\mathcal{D}}, \hat{\mathcal{D}}, \hat{\mathcal{R}}$ の包含関係は、木庭ほか²⁵⁾に、後者の定義のもとでの $\hat{\mathcal{D}}, \hat{\mathcal{D}}, \hat{\mathcal{R}}$ の包含関係は、Brzozowski, Muro⁹⁾ に示されている。両者を比較した場合、前者では $\hat{\mathcal{D}} = \hat{\mathcal{R}}$ となるのに、後者では $\hat{\mathcal{D}} \supsetneq \hat{\mathcal{R}}$ となり、異なった包含関係が得られる。

Ibaraki, Kameda¹⁷⁾ は、前章で論じた DITS の概念が多版モデルに対しても有効に拡張できることを示している。さらに単版モデルで定義したのと同様に、 $\hat{\mathcal{R}}$ の部分クラス、MRW, MWW, MRR, およびその組み合わせからできる部分クラスの包含関係を単版モデルでのクラスとの包含関係も含めて明らかにし、各クラスの所属問題の判定アルゴリズムの計算の複雑度を明らかにした¹⁷⁾。その内で特に興味深い結果として次の三つを挙げておこう。(1) Papadimitriou, Kanellakis³²⁾ は、所属問題が多項式時間のクラスとして DMVSR (DSR の多版モデルへの拡張クラスと考えてよい) というクラスを提案しているが、この DMVSR が本質的にはクラス MWW と同一のものであることを示していること。(2) 制約付2ステップモデルでは、 $\hat{\mathcal{R}} = \text{MWW}$ となり、単版モデルでの結果と対応していること。(3) 単版モデルでは、RW の所属問題は NP-完全であった (表-3 参照) のに、MRW の所属問題は多項式時間のアルゴリズムをもつことが証明されている。このことは版の選択で判定問題がより難しくなるであろうと考えるのが自然でないことを示唆している。

多版モデルでは任意の数の版を作れることを許しているが、許される版数と同時処理能力との関係も研究されている^{25), 32)}。スケジュール e を実行する際に、各データ項目ごとに高々 k 版 (新しいものから k 版とは限らない) しかコピーを作れないという制約のもとで、 e を $*$ -直列可能 (ただし $*$ = δ, τ, σ) にできるとき、 e は k 版 $*$ -直列可能といい、そのクラスを各々、 $\mathcal{D}^{(k)}, \mathcal{D}^{(k)}, \mathcal{R}^{(k)}$ と記す。Papadimitriou, Kanellakis³²⁾ は $\mathcal{D}^{(k)}$ が k に関して無限の階層をもつこと

を、木庭ほか²⁵⁾は $\mathcal{G}^{(k)}, \mathcal{R}^{(k)}$ も k に関して無限の階層性を有すること、さらに三つのトランザクションモデルの各々の場合において、 $\mathcal{D}^{(k)}, \mathcal{G}^{(k)}, \mathcal{R}^{(k)}$ ($k=1, 2, \dots$) が k に関してどのような包含関係をもつかを明らかにしている。

故障回復のためのコピー以外にコピーを複数個もつ多版モデルが、現実にとりわけ意味をもつモデルなのかは議論の余地があろう。たとえば、一旦書き込んだら以後更新ができず、読み込みのみが可能であるような光ディスクを記憶媒体として用いるシステムなどでは自然と版が作られていき、多版モデルが実現しやすと思われる。実際に多版モデルを用いたデータベースシステムの初期のものとしては、Honeywell 社の FMS システム* が知られており、1973 年には多版モデルがすでに実用化されていた。最近のものとしては、米国 Computer Corporation of America の分散データベースシステム DDM がよく知られており¹²⁾、多版モデルに 2 相ロック機構⁴⁴⁾を組み合わせた方式がとられている。

7. む す び

本稿を終えるにあたり、まず、以上で紹介した、与えられたスケジュールの直列可能性についての理論的な研究の意味を、現実のシステムとの対応から改めて考えてみよう。スケジュールとは複数のデータベース利用者からの Read と Write を到着順に並べたものであり、各利用者は他の利用者がデータベースに対してどのようなアクセス要求をしてくるかまったくおかないしに、自分がシステムの唯一の利用者であるという想定で、プログラムの実行に伴った Read, Write をデータベース管理システムに送ってくると考えられる(利用者の同時処理制御に関する透明性)。したがって、本稿で紹介した研究は、各利用者からなんの制御もなく送られてきた Read, Write より構成されたスケジュールのうち、単版、多版の各モデルのもとで、データベースの首尾一貫性を保ちながら実行できるものの特性づけを目的としたものである。その意味でまがききでもお断りしたように、本稿では、入力系列に制約を設けたり(たとえば、2 相ロック (two phase lock) 機構¹³⁾)、スケジューラが各 Read, Write に付せられたタイムスタンプを用いて Read, Write の実行順序を制御したり^{41), 42)}して、いかに直列可能なスケ

ジュールを実現したらよいかという議論は一切含んでいない。

直列可能性の最近の話題として先読スケジュール (coutious schedule) の研究がある(たとえば、文献 7), 10), 11), 18), 22), 23))。先読スケジュールでは、トランザクションの先頭の Read または Write がスケジューラに受けとられたとき、そのトランザクションで将来実行される Read と Write の操作の集合がわかる(ただし、いつ来るかはわからない)という仮定がなされる。それらの将来の Read, Write も考慮したうえで、もし、現在入力された Read または Write を実行すると直列可能な実行系列ができなくなる可能性が少しでもある場合には、その実行を遅らせて次の操作の実行を試みる。このようにして、一旦スケジューラに受け取られたトランザクションの Read, Write は、途中で放棄されることなく、必ず直列可能なスケジュールに並びかえられて実行されるようにするものである。利用者にとっては、途中までトランザクションが実行されたにもかかわらず、直列可能なスケジュールが作れないということで途中でトランザクションの実行を放棄されるよりも、待たされる可能性はあるにしても、とにかく実行してもらえんということがわかっていれば安心できる。ただし、将来の Read, Write の集合がスケジューラに知られるという仮定の現実性および先読スケジュールを実現するためのデータベース管理システムの(計算の複雑度の理論では測り知れない実際の)オーバーヘッドについては検討してみる必要がある。

一昨年、イタリアのフィレンツェで開催された第 9 回 Very Large Data Base (VLDB) 国際会議(データベースの分野ではよく知られた会議である)に、筆者は論文発表のため参加する機会を得た。この会議の「Concurrency Control: Are we done with algorithms?」と題するパネル討論では Papadimitriou が司会を務め、Silberschatz, Schlageter など、同時処理制御に関する著名な研究者をパネラとして、活発な討論がなされた。そこで非常に興味深かったことは、現在の直列可能性、さらに同時処理制御の理論的研究に疑問を投げかける研究者が多かったことである。今までにたくさんの直列可能性を実現するアルゴリズムが提案されてきた(ある発言者は、「百人の学生がいれば各人別々のアルゴリズムを提案できる」とまで言っていた)にもかかわらず、現実に使われているのはほとんどが 2 相ロック機構であるという事実が研究の

* Honeywell File Management Supervisor, Order Number DB 54, Honeywell Information Systems Inc., 1973.

応用性に疑問を抱かせる根拠のようであった。今後は新しいアルゴリズムの提案をすることより、システムの各状態下において、今までに提案されたアルゴリズムのどれが適しているのかを探ることの方が重要であり、アルゴリズムの性能評価の時代に入ったことは参加者の皆が認めるところであった。現にこの方面の研究が盛んになりつつある。

謝辞 末筆ながら、日頃、ご指導と温かい励ましを頂いている本学長谷川利治教授に衷心より感謝の意を表す。本稿を書き上げるにあたっては、カナダ国ウォータールー大学 Brzozowski 教授との熱のこもった討論、ならびに、豊橋技術科学大学茨木俊秀教授および神戸商科大学加藤直樹助教授と機会あるごとに討論をして頂いたことが非常に役立った。ここに深謝の意を表す。最後に、筆者がデータベースの研究を始めるきっかけを与えて下さったカナダ国サイモンフレーザー大学亀田恒彦教授および研究を進展させる上でいろいろお教え頂いた米国オレゴン州立大学箕浦敏美助教授に心より感謝する。

参 考 文 献

- 1) Bayer, R., Heller, H. and Reiser, A.: *Parallelism and Recovery in Database Systems*, ACM Trans. Database Syst., Vol. 5, No. 2, pp. 139-156 (1980).
- 2) Bayer, R., Elhardt, K., Heller, H. and Reiser, A.: *Distributed Concurrency Control in Database Systems*, Proc. 6th Int. Conf. on VLDB, pp. 275-284 (1980).
- 3) Bernstein, P. A., Shipman, D. W. and Wong, W. S.: *Formal Aspects of Serializability in Database Concurrency Control*, IEEE Trans. Softw. Eng., Vol. SE-5, No. 3, pp. 203-215 (1979).
- 4) Bernstein, P. A. and Goodman, N.: *Time-stamp-based Algorithms for Concurrency Control in Distributed Database Systems*, Proc. 6th Int. Conf. on VLDB, pp. 285-300 (1980).
- 5) Bernstein, P. A. and Goodman, N.: *Concurrency Control in Distributed Database Systems*, ACM Comput. Surv., Vol. 13, No. 2, pp. 185-222 (1981).
- 6) Bernstein, P. A. and Goodman, N.: *Multiversion Concurrency Control-Theory and Algorithms*, ACM Trans. Database Syst., Vol. 8, No. 4, pp. 465-483 (1983).
- 7) Buckley, G. N. and Silberschatz, A.: *Obtaining Progressive Protocols for a Simple Multiversion Database Model*, Proc. 9th Int. Conf. on VLDB, VLDB Endowment, pp. 74-80 (1983).
- 8) Brzozowski, J. A.: *On Models of Transactions*, Tech. Rep. # 84001, Dept. of Applied Math. and Physics, Faculty of Engineering, Kyoto University (1984).
- 9) Brzozowski, J. A. and Muro, S.: *On Serializability*, Tech. Rep. # 840012, Dept. of Applied Math. and Physics, Faculty of Engineering, Kyoto University (1984).
- 10) Casanova, M. A. and Bernstein, P. A.: *General Purpose Schedulers for Database Systems*, ACTA Informatica, Vol. 14, pp. 195-220 (1980).
- 11) Casanova, M. A.: *The Concurrency Control Problem for Database Systems*, Lecture Notes in Computer Science, 116, Springer-Verlag, Berlin (1981).
- 12) DuBourdieu, D. J.: *Implementation of Distributed Transactions*, Proc. 6th Berkeley Workshop on Dist. Data Manag. and Comput. Networks, pp. 81-94 (1982).
- 13) Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L.: *The Notions of Consistency and Predicate Locks in a Database System*, Comm. ACM, Vol. 19, No. 11, pp. 624-633 (1976).
- 14) Garey, M. R. and Johnson, D. S.: *Computers and Intractability-A Guide to the Theory of NP-Completeness*, Freeman, San Francisco (1979).
- 15) Gray, J. N.: *Notes on Data Base Operating Systems*, Lecture Notes in Computer Science, 60, Springer-Verlag, Berlin, pp. 393-481 (1978).
- 16) Ibaraki, T., Kameda, T. and Minoura, T.: *Serializability with Constraints: DITS and Its Applications*, TR 82-12, Dept. of Comput. Sci., Simon Fraser University (1982). (Extended abstract of this report: *Disjoint-Interval Topological Sort: A Useful Concept in Serializability Theory*, Proc. 9th Int. Conf. on VLDB, VLDB Endowment, pp. 89-91 (1983); (邦文) 茨木, 亀田, 箕浦: Serializable Class の構造について, 京大数解研講究録 494, pp. 102-113 (1983).)
- 17) Ibaraki, T. and Kameda, T.: *Multi-Version vs. Single-Version Serializability*, LCCR TR 83-1 and CMPT TR 83-14, Lab. for Comp. and Commun. Research, Simon Fraser University (1983).
- 18) Ibaraki, T., Kameda, T. and Katoh, N.: *Cautious Schedulers for Database Concurrency Control*, TR 85-2, Dept. of Comput. Sci., Simon Fraser University (1985).
- 19) 上林弥彦: 一データベースの基礎理論(6)ー共有データベースの諸問題に対する理論, 情報処

- 理, Vol. 24, No. 8, pp. 1020-1037 (1983).
- 20) 亀田恒彦: 分散形データベースについて, システムと制御, Vol. 25, No. 2, pp. 67-73 (1981).
- 21) Kaneko, A., Nishihara, Y., Tsuruoka, K. and Hattori, M.: *Logical Clock Synchronization Method for Duplicated Database Control*, Proc. 1st Int. Conf. on Distributed Computing Systems, pp. 601-611 (1979).
- 22) Katoh, N., Ibaraki, T. and Kameda, T.: *Cautious Transaction Schedulers with Admission Control*, ACM Trans. Database Syst., Vol. 10, No. 2, pp. 205-229 (1985).
- 23) Katoh, N., Kameda, T. and Ibaraki, T.: *A Cautious Scheduler for Multi-step Transactions*, TR 85-1, Dept. of Comput. Sci., Simon Fraser University (1985).
- 24) Kedem, Z. and Silberschatz, A.: *Controlling Concurrency Using Locking Protocols*, Proc. IEEE 20th Symp. Foundations of Comp. Sci., pp. 274-284 (1979).
- 25) 木庭, 室, 長谷川: データベースシステムの同時処理制御における直列可能性のいくつかのクラスについて, 京大数解研講究録 556, pp. 59-70 (1985).
- 26) Lausen, G.: *Serializability Problems of Interleaved Transactions*, Trends in Information Processing Systems, Lecture Notes in Computer Science, 123, Springer-Verlag, Berlin, pp. 252-265 (1981).
- 27) Minoura, T. and Wiederhold, G.: *Resilient Extended True-copy Token Scheme for Distributed Database System*, IEEE Trans. Softw. Eng., Vol. SE-8, No. 3, pp. 173-189 (1982).
- 28) 箕浦敏美: private communication (1984).
- 29) Muro, S., Kameda, T. and Minoura, T.: *Multi-version Concurrency Control Scheme for a Database System*, J. Comput. Syst. Sci., Vol. 29, No. 2 pp. 207-224 (1984).
- 30) Muro, S., Mizutani, T. and Hasegawa, T.: *Multiversion Concurrency Control Scheme for a Distributed Database System - A Trial to Break Concurrent Update of Redundant Copies*, Tech. Rep. #84021, Dept. of Applied Math. and Physics, Faculty of Engineering, Kyoto University (1984). (京大数解研講究録 547, pp. 79-111 (1985).)
- 31) Papadimitriou, C.H.: *The Serializability of Concurrent Database Updates*, J. ACM, Vol. 26, No. 4, pp. 631-653 (1979).
- 32) Papadimitriou, C.H. and Kanellakis, P.C.: *On Concurrency Control by Multiple Versions*, ACM Trans. Database Syst., Vol. 9, No. 1, pp. 89-99 (1984).
- 33) Reed, D.P.: *Implementing Atomic Actions on Decentralized Data*, ACM Trans. Computer Systems, Vol. 1, No. 1, pp. 3-23 (1983).
- 34) Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M., II: *System Level Concurrency Control for Distributed Data Base Systems*, ACM Trans. Database Syst., Vol. 3, No. 2, pp. 178-198 (1978).
- 35) Rosenkrantz, D.J., Stearns, R.E. and Lewis, P.M., II: *Consistency and Serializability in Concurrent Database Systems*, SIAM J. Comput., Vol. 13, No. 3, pp. 508-530 (1984).
- 36) Sethi, R.: *A Model of Concurrent Database Transactions*, Proc. 22nd IEEE Symp. Foundations of Comp. Sci., pp. 175-184 (1981).
- 37) Sethi, R.: *Useless Actions Make a Difference: Strict Serializability of Database Updates*, J. ACM, Vol. 29, No. 2, pp. 394-403 (1982).
- 38) Sethi, R.: *Pebble Games for Studying Storage Sharing*, Theor. Comput. Sci., Vol. 19, No. 3, pp. 69-84 (1982).
- 39) Silberschatz, A. and Kedem, Z.: *Consistency in Hierarchical Database Systems*, J. ACM, Vol. 27, No. 1, pp. 72-80 (1980).
- 40) Stearns, R.E., Lewis, P.M. II and Rosenkrantz, D.J.: *Concurrency Control for Database Systems*, Proc. 17th IEEE Symp. Foundations of Comp. Sci., pp. 19-32 (1976).
- 41) Stearns, R.E. and Rosenkrantz, D.J.: *Distributed Database Concurrency Using Before-values*, Proc. ACM-SIGMOD, pp. 74-83 (1981).
- 42) Thomas, R.H.: *A Solution to the Concurrency Control Problem for Multiple Copy Databases*, digest of papers IEEE COMPCON spring, pp. 56-62 (1978).
- 43) Thomas, R.H.: *A Majority Consensus Approach to Concurrency Control*, ACM Trans. Database Syst., Vol. 4, No. 2, pp. 180-219 (1979).
- 44) Ullman, J.D.: *Principles of Database Systems. (Second Edition)*, Computer Science Press, Potomac, Maryland, pp. 369-408 (1982).
- 45) Yannakakis, M.: *Serializability by Locking*, J. ACM, Vol. 31, No. 2, pp. 227-244 (1984).

(昭和60年2月18日受付)