適応的分散アルゴリズム 第６章
無待機システム

川染翔吾

6.2 無待機性

無待機性

- 線形化可能性を満たす共有オブジェクトを実現するための簡単な方法は相互排除
  - 共有オブジェクトを占有するプロセスが停止したとき、分散システム全体が停止してしまう
- **無待機性**：他のプロセスの停止故障や動作速度にかかわらず、プロセス自信の作業を有限時間内に完了できる
  - n プロセスからなるシステムで $n-1$ 個のプロセスが停止故障しても、残り 1 個のプロセスは正常に動作する

6.3 共有オブジェクトを実現する無待機アルゴリズム

共有レジスタ

- **原子レジスタ**：複数のプロセスが並行して実行したときでも、その大域履歴が線形化可能性を保証するレジスタ
- 共有レジスタは、格納できる値の種類数、$Read$ 命令を実行できるプロセス数、および $Write$ 命令を実行できるプロセス数の違いで分類される

レジスタの種類

- 格納できる値が 0 から $k-1$ までの k 種類のとき k 値レジスタ
  - 最も単純なものは 2 値レジスタ
  - k が 3 以上なら多値レジスタ
- r 個のプロセスが $Read$ 命令を実行可能で、w 個のプロセスが $Write$ 命令を実行可能なとき rRwW レジスタという
  - 最も単純なものは 1R1W レジスタ
  - 単一か複数かだけを表したいときは、SRSW レジスタ、MRSW レジスタ、SRMW レジスタ、MRMW レジスタ
- 最も単純なレジスタは SRSW の 2 値レジスタ

高機能なレジスタの実現

- SRSW の 2 値レジスタは原子レジスタとして動作するものとして進める
- 任意の r, w, k に対して、1R1Wの 2 値レジスタを用いて rRwW の k 値レジスタを構成する無待機アルゴリズムを段階的に考える

1. SRSW の 2 値レジスタを用いて、SRSWの多値レジスタを構成する無待機アルゴリズム
2. SRSW の多値レジスタを用いて、MRSWの多値レジスタを構成する無待機アルゴリズム
3. MRSWの多値レジスタを用いて、MRMWの多値レジスタを構成する無待機アルゴリズム

2 値レジスタから多値レジスタの構成

- SRSW の 2 値レジスタを用いて、SRSWの多値レジスタを構成する
- 線形化可能性は局所性が成り立つので、SRSW の k 値レジスタを一つ構成することを考える

SRSW-M

- k 値レジスタ R を実現する
- k 個の 2 値レジスタ $R_0, R_1, \ldots, R_{k-1}$ を使用する
- $R = i$ を $R_i = 1, R_0 = R_1 = \cdots = R_{i-1} = 0$ として実現する
  - 値 1 を格納する 2 値レジスタの最小インデックスが k 値レジスタ R が格納している値
- R は初期値としてある値 $v_0 (0 \leq v_0 \leq k-1)$ を持ち、$R_{v_0} = 1, R_i = 0 (i \neq v_0)$ が成り立つ

単純な読出し方法で失敗する例

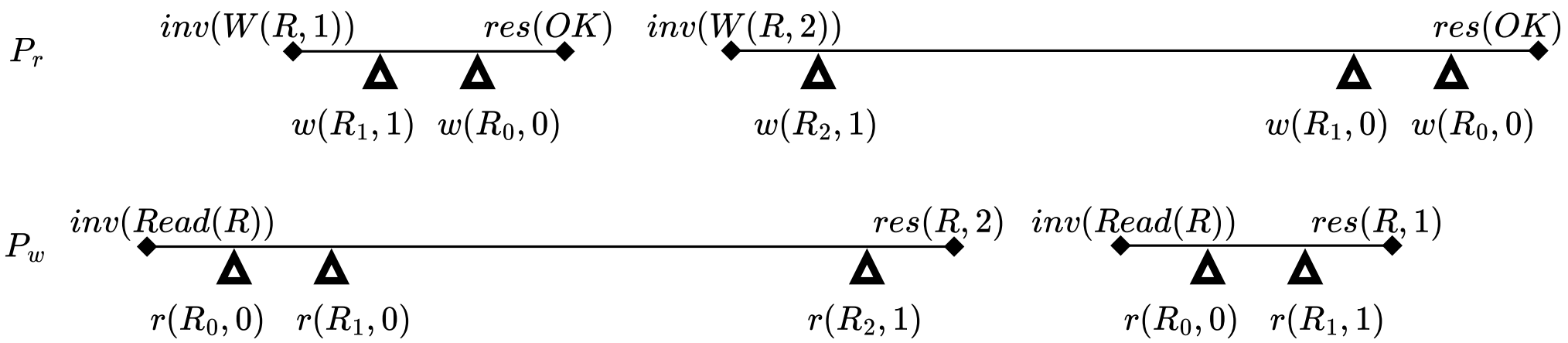
書込み

- R_v に 1 を書込み、$R_i (0 \leq i \leq v-1)$ にインデックスの降順で 0 を書き込む

読出し

- R_0 からインデックスの昇順に読み出し、最初に 1 を読み出したレジスタのインデックスを R から読み出した値とする

単純な読出し方法で失敗する例



SRSW-M

読出し動作

1. $i \leftarrow 0$
2. `while` $R_i = 0$ `do`
    - $i \leftarrow i + 1$
3. $value \leftarrow i$
4. `for` $j \leftarrow i - 1$ `downto` 0 `do`
    - `if` $R_j = 1$ `then` $value \leftarrow j$
5. `return` $value$

値 v の書込み動作

1. $R_v \leftarrow 1$
2. `for` $i \leftarrow v - 1$ `downto` 0 `do`
    - $R_i \leftarrow 0$
3. `return` (OK)

SRSW-M

SRSW-M は k 個の SRSW の 2 値レジスタを用いて、SRSW の k 値レジスタを実現する無待機アルゴリズムである

証明

無待機性を示す
2 の `while` が終了する、つまりレジスタ R_i から値 1 を読み出すことを示す

初期状態では $R_{v_0} = 1$ が成り立つ。
2 値レジスタ R_i に値 0 を書き込むとき、その前に $R_v(v > i)$ に値 1 を書き込んでいる。このことから P_r がレジスタ R_i を読み出すときにはあるレジスタ $R_j(j \leq i)$ が値 1 を保持していることを示せる

SRSW レジスタから MRSW レジスタの構成

- SRSW の多値レジスタを用いて、MRSWの多値レジスタを構成する
- レジスタ R に対して $READ$ 命令を実行できる n 個のプロセスを $P_r(1 \leq r \leq n)$、
  $READ$ 命令を実行できる単一のプロセスを Q_w と表す
- MRSW レジスタ R を実現するために $n^2 + n$ 個の SRSW レジスタを使用

- Val_r：プロセス Q_w が共有レジスタ R に書き込んだ値を各プロセス P_r に伝えるためのレジスタ
- $Report_{i,j}$：プロセス P_i がレジスタ R から読み出した値を他のプロセス P_j に伝えるためのレジスタ
- Val_r および $Report_{i,j}$ には、レジスタ R に格納する値とプロセス Q_w が R にその値を書き込んだときの時刻印の組を格納する
- 全ての SRSW レジスタの初期値は、R の初期値 v_0 と時刻印の初期値 0 の組 $(v_0, 0)$

単純な読出し方法で失敗する例

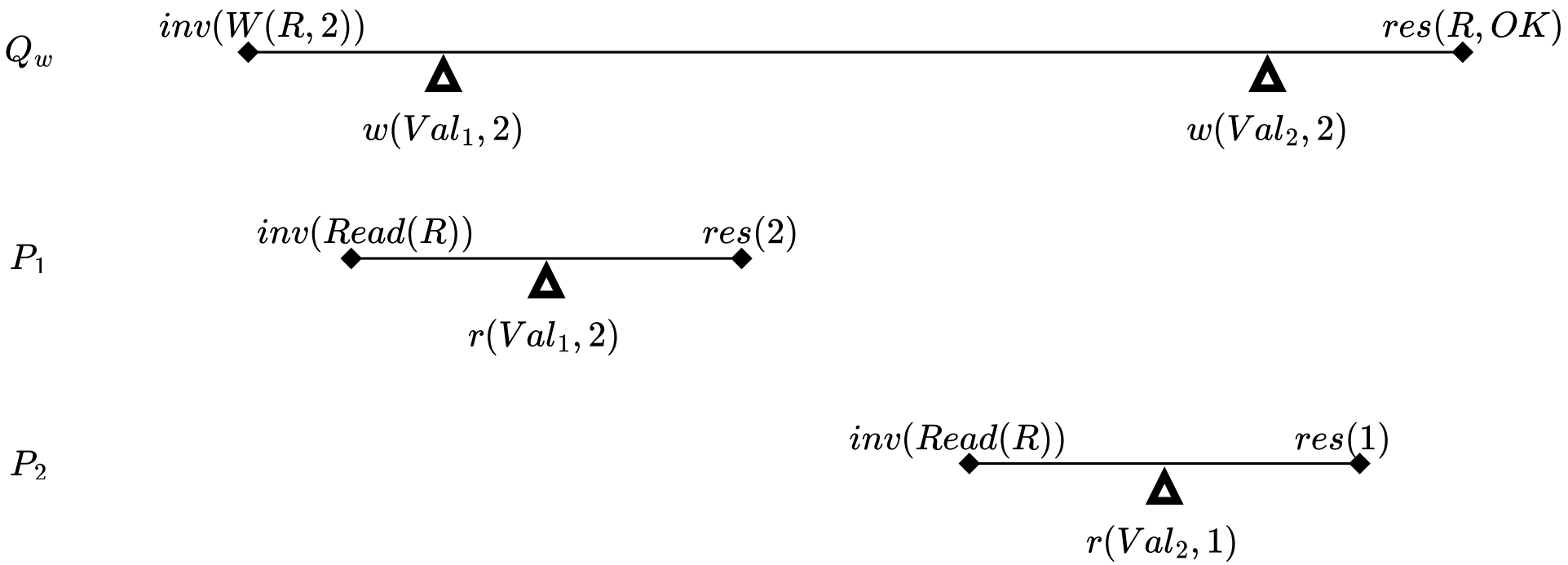
書込み

- $Val_i (0 \leq i \leq n-1)$ にインデックスの昇順で書き込む

P_i による読出し

- Val_i から読み出す

単純な読出し方法で失敗する例



MRSW-M

P_r の読出し動作

1. $(v[0], ts[0]) \leftarrow Val_r$
2. `for` $i \leftarrow 1$ `to` n `do`
    - $(v[i], ts[i]) \leftarrow Report_{i,r}$
3. $j = \mathrm{maxarg}_{0 \leq i \leq n} ts[i]$
4. `for` $i \leftarrow 1$ `to` n `do`
    - $Report_{r,i} \leftarrow (v[j], ts[j])$
5. `return` $v[j]$

値 v の書込み動作

1. $ts \leftarrow ts + 1$
2. `for` $i \leftarrow 1$ `to` n `do`
    - $Val_i \leftarrow (v, ts)$
3. `return` (OK)

MRSW レジスタから MRMW レジスタの構成

- MRSW の多値レジスタを用いて、MRMWの多値レジスタを構成する
- 構成する nRmW レジスタ R に対して $Read$ 命令を実行できる n 個のプロセスを P_r とし、$Write$ 命令を実行できる m 個のプロセスを Q_w とする
- m 個の $(n+m)$R1W レジスタ $R_i(1 \leq i \leq m)$ を使用する
- 共有レジスタ R_i に対しては、プロセス Q_i が $write$ 命令を実行でき、各プロセス $P_r(1 \leq r \leq n)$ および $Q_w(1 \leq w \leq m)$ が $read$ 命令を実行できる
- レジスタ R_i が格納される値は、レジスタ R が格納するデータと、プロセス Q_i が R_i にそのデータを書き込んだときの時刻印
- 全ての MRSW レジスタ の初期値は、レジスタ R の初期値 v_0 と時刻印の初期値 0 の組 $(v_0, 0)$

MRMW-M

P_r による読出し動作

1. `for` $i \leftarrow 1$ `to` m `do`
    - $(v[i], ts[i]) \leftarrow R_i$
2. $j \leftarrow \mathrm{maxarg}_{1 \leq i \leq m} ts[i]$
    - $ts[i]$ が同じ場合は最小の i
3. `return` $v[j]$

Q_w による値 v の書込み動作

1. `for` $i \leftarrow 1$ `to` m `do`
    - $ts[i] \leftarrow R_i.ts$
2. $ts \leftarrow \max\{ts[i] | 1 \leq i \leq m\} + 1$
3. $R_w \leftarrow (v, ts)$
4. `return` (OK)