Теория и практика многопоточного программирования

ЛЕКЦИЯ 8. КОНСЕНСУС

В прошлый раз...

Свойства согласованности и регистры

История

Линеаризация истории

Темы лекции

Консенсус

Число консенсуса

Протокол, состояние протокола

Валентность состояния

Создание консенсуса из примитивов

Консенсус

Консенсус – совместное однократное принятие общего решения N потоками из предложенных.

Формально:

```
interface IConsensus<T> {
      T decide(T val);
}
```

Объект консенсуса – реализация интерфейса, при которой если в каждом из заинтересованных потоков произойдёт вызов метода `decide`() не более, чем один раз, то метод во всех случаях вернёт одинаковое значение, и это значение будет одним из переданных в вызов.

Про консенсус

Задача N-поточного консенсуса – создание объекта консенсуса для системы N потоков.

Некоторый класс ***C*** **решает задачу N-поточного консенсуса**, если существует **протокол** достижения консенсуса изпользующий произвольное количество экземпляров класса ***C*** и атомарных регистров.

Числом консенсуса для класса ***C*** называется максимальное число потоков, для которого этот класс решает задачу консенсуса.

Следствие: Предположим, можно *воспроизвести* класс ***C*** из конечного числа экземпляров класса ***D*** и атомарных регистров. Тогда если класс ***C*** решает задачу консенсуса для N потоков, то и класс ***D*** также решает эту задачу.

Бинарный (T == bool) консенсус с числом 2

T = {0, 1} – набор допустимых значений

N = 2 – число потоков (*A* и *B*)

Протокол состоит из «ходов» (moves). Ход – вызов методов некоторого разделяемого объекта.

Состояние протокола = состояние разделяемых объектов и потоков

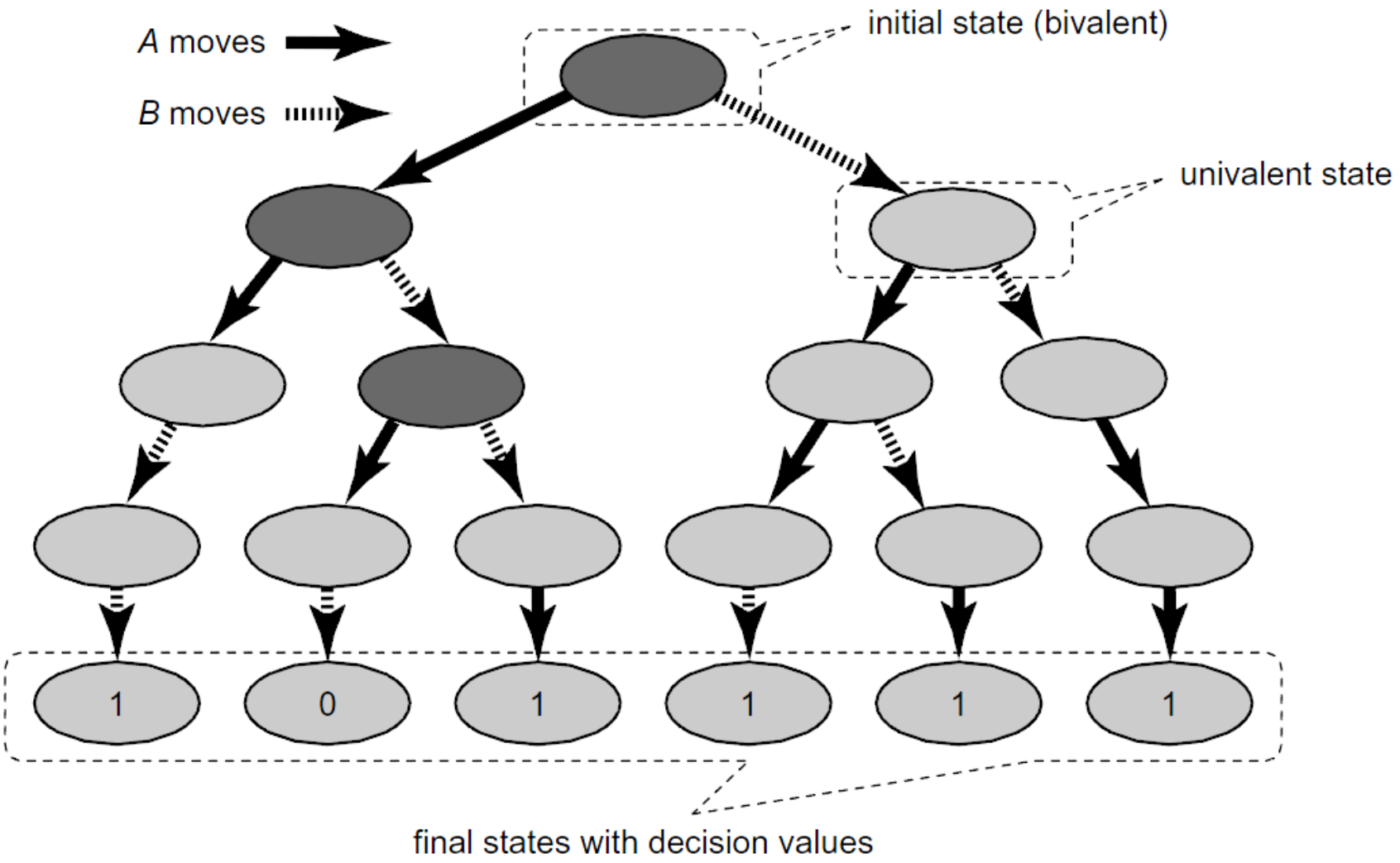
Начальное состояние – состояние до совершения первого хода всеми потоками

Конечное состояние – состояние после завершения вызова из всех потоков

Бивалентное состояние – состояние, для которого решение ещё не фиксировано. Существуют допустимые последовательности ходов, приводящие к различным решениям

Унивалентное состояние – состояние, все исходы из которого приводят к одному решению

Дерево состояний



Леммы

Лемма: любой 2-поточный протокол консенсуса имеет бивалентное начальное состояние.

Лемма: любой N-поточный протокол консенсуса имеет бивалентное начальное состояние.

Состояние является **критическим**, если оно бивалентно, и ход любого потока меняет состояние протокола на унивалентное.

Лемма: любой неблокирующий протокол консенсуса имеет критические состояния.

Консенсус vs Atomic Register

При помощи атомарных регистров НЕЛЬЗЯ решить задачу консенсуса.

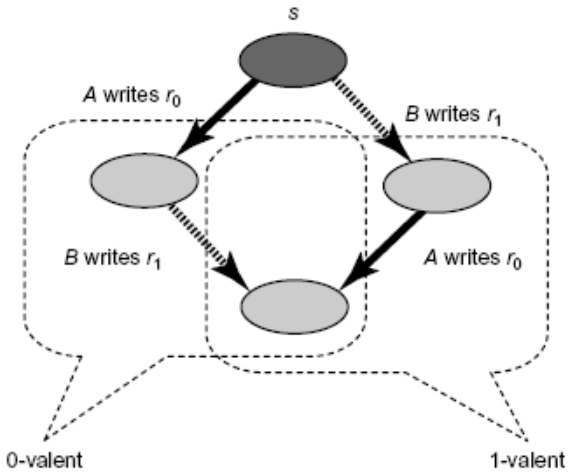
Теорема: число консенсуса атомарного регистра равно 1.

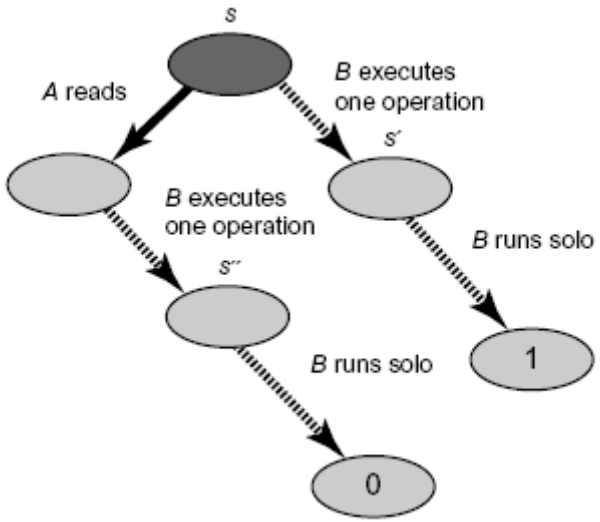
- Возможные выходы из критического состояний:
  - Чтение одного регистра
  - Запись разных регистров
  - Запись одного регистра

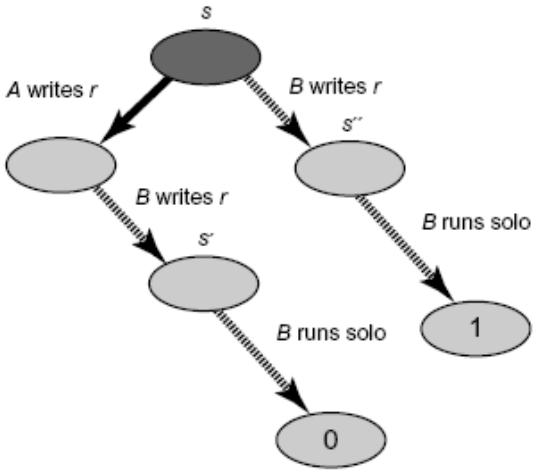
Следствие: невозможно построить протокол консенсуса с использованием атомарных регистров.

NB: нельзя построить неблокирующие разделяемые структуры данных, используя только операции чтения и записи

Иллюстрации к доказательству







Протокол консенсуса

```
public abstract class ConsensusProtocol<T> implements Consensus<T> {
  protected T[] proposed = (T[]) new Object[N];
  // announce my input value to the other threads
  void propose(T value) {
    proposed[ThreadID.get()] = value;
  }
  // figure out which thread was first
  abstract public T decide(T value);
}
```

Строим объект консенсуса

Queue / Stack

Multiple assignment objects

RMW

Common2 RMW

CAS

Queue / Stack

Теорема: Очередь с 2 читателями имеет число консенсуса минимум 2.

- Если создать линеаризуемую очередь на 2 читателя и инициализировать её значениями [1, 0], то результат консенсуса вычисляется как
  - `proposed[my] = VALUE`
  - `queue.dequeue() ? proposed[my] : proposed[his].`

Следствие: любая очередь имеет число консенсуса 2.

Multiple-assignment object

***(m,n)*-assignment object** – объекты, содержащие **n** полей и атомарно присваивающие **m** из них.

Теорема: не существует неожидающей (wait-free) реализации такого рода (m,n)-объектов, построенной на атомарных регистрах.

```
abstract class Multiple23 {
        T[] values = new T[3];

        abstract void Assign<T>(int idx1, T item1, int idx2, T item2);

        T Get(int i) {
                return values[i];
        }
}
```

Доказательство теоремы

Достаточно показать, что можно построить 2-консенсус реализацию, имея (2,3)-объект.

Пусть массив проинициализирован **null**.

A: obj.Assign(0, a, 1, a);

B: obj.Assign(1, b, 2, b);

A: [a, a, ***null***] (A ? **A**), [a, ***b***, ***b***] (A ? **A**), [a, ***a***, ***b***] (A ? **B**)

B: [***null***, b, b] (B ? **B**), [***a***, ***b***, b] (B ? **A**), [***a***, ***a***, b] (B ? **B**)

```
var nullable = obj.read((TID()+2) % 3);
if (nullable == null || nullable == obj.read(1))
        return my;
else

        return his;
```

Теорема 2

Th: *(n, n*(n+1) / 2)*-объект имеет число консенсуса как минимум n.

Введём нумерацию.

Для **n** полей r_i – для эксклюзивной записи *i*-го потока

Для остальных r_{ij} – для пересекающейся записи 2 потоков $i > j$

Протокол:

Для всех возможных пар потоков установить порядок (аналогично (2,3)) по регистрам r_i, r_j, r_{ij}

RMW (Read-Modify-Write)

Класс регистров, определяемых некоторой функцией *f(v)*

```
atomic T RMW(ref T v, Func f) {
        var tmp = v;
        v = f(v);
        return tmp;
}
```

Примеры:

$RMW_{=}$ = простое чтение. *«Тривиальный RMW»*

RMW_{CONST}= get-and-set

RMW_{++} = atomic increment

$RMW_{+=}$ = atomic add

RMW_{CAS} - не совсем RMW. $f_{expected,new}(v)$ = (v == expected) ? new : v;

Теорема

Нетривиальный RMW-регистр имеет число консенсуса **минимум 2.**

Доказательство через построение.

Задание: построить 2-консенсус на базе нетривиального RMW

Подсказки:

- 1) выбрать *f(v), v0* такие, что *f(v0) != v0*
- 2) опираться на реализацию 2-очереди

Следствие: RMW-регистр нельзя собрать из атомарных

Common2 RMW

Common2 – класс регистров (примитивов), использовавшихся для синхронизации в конце XX века.

Common2-функции f(x):

Все $f_i(v)$ *из* F принадлежат классу Common2, если удовлетворяют **одному из** условий:

- $f_j(f_i(v)) = f_i(f_j(v))$ - коммутативны
- $f_i(f_j(v)) = f_i(v)$ - перезаписывают

Число консенсуса любого Common2-регистра равно 2.

Краткое доказательство: из-за свойств по определению, третий поток не в состоянии сказать, кто из 2 других потоков был первым, и оба ли выполнились.

Compare-and-set/swap

Теорема: регистр, реализующий CAS, обладает числом консенсуса бесконечность.

Доказательство:
через построение (sticky byte)

Выводы

Консенсус требует особых примитивов

Объекты с одинаковым числом консенсуса взаимно реализуемы

CAS – это панацея синхронизации