Теория и практика многопоточного программирования

Лекция 6

Неганов Алексей

Московский физико-технический институт (национальный исследовательский университет)
Кафедра теоретической и прикладной информатики

Москва 2020

Формализм конечного автомата

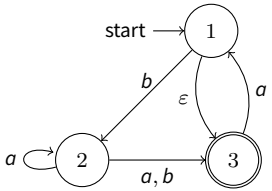
Определение

Недетерминированный конечный автомат (НКА) — это пятёрка

$$M = (Q, \Sigma, \delta, q_0, F), \tag{1}$$

где Q — конечное множество состояний,
Σ — конечное множество допустимых входных символов (входной алфавит),
ε — пустая цепочка символов,
$\delta\colon\ Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ — функция переходов, отображающая пару «состояние – символ» в некое подмножество Q,
$q_0 \in Q$ — начальное состояние,
$F \subseteq Q$ — множество заключительных (допускающих) состояний.

Формализм конечного автомата



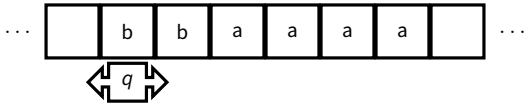


Рисунок 1 – Диаграмма состояний недетерминированного конечного автомата

Формализм конечного автомата: модель потока

- Q — состояние памяти и регистров процессора,
- q_0 — точка начала исполнения потока *(entry point)*,
- Переход между состояниями (q_i, q_j) — событие *(event)*, причём каждому событию ставится в соответствие точка на временной шкале,
- Все события по определению одномоментны (квантование времени).
- Никакие два события не являются одновременными.
- События a и b называются **упорядоченными,** если a предшествует b: $a \to b$.
- Произвольные два события a и b могут быть неупорядочены: $a \not\to b$ и $b \not\to a$.
- Отношение порядка транзитивно: если $a \to b$ и $b \to c$, то $a \to c$.
- **Интервал** $I(a, b)$ — это пара событий (a, b) такая, что $a \to b$.
- Интервал $I(a, b)$ **предшествует** интервалу $I(b, c)$ ($I(a, b) \to I(b, c)$), если $b \to c$.
- Интервалы, не связанные отношением «$\to$», называются **соисполняемыми** *(concurrent)*.

Пример простого доказательства корректности для lock-free hash table
`https://docs.huihoo.com/javaone/2007/java-se/TS-2862.pdf`
Ключевые слова для желающих углубиться: discrete event simulation, communicating finite-state machines

Формализм конечного автомата: модель потока

```
#include <thread>
#include <atomic>
#include <stdio.h>
#include <vector>

std::atomic<int> a, b; // state: variables

void f1() {
    a = 0xF; // event: variable assignment
    while (a != b); // event: wait for condition
    printf("OK\n");
}

void f2(int c) {
    b |= c; // event: variable assignment
}

int main() {
    b.store(0);
    std::vector<std::thread> v(5);
    v[4] = std::thread(f1);
    for (int i = 0; i < 4; i++)
        v[i] = std::thread(f2, 1 << i);
    for (auto &t: v)
        t.join();
    printf("That's all\n");
}
```

Определение

Назовём интервал CS_i **критической секцией**, если для детерминированности совместного исполнения потоков необходимо, чтобы для любых потоков A, B их критические секции были упорядочены: $CS_A \to CS_B$ или $CS_B \to CS_A$. Выполнение этого условия для системы потоков называется свойством взаимоисключения *(mutual exclusion)*.

Критерий Бернстайна

Пусть $R(P_i)$ множество переменных, значение которых поток P_i использует в операциях чтения, $W(P_i)$ — множество переменных, использующихся в операциях записи. Тогда совокупное исполнение потоков P_1 и P_2 детерминировано, если

$$\begin{cases} W(P_1) \cap W(P_2) = \varnothing \\ R(P1) \cap W(P_2) = \varnothing \\ W(P_1) \cap R(P_2) = \varnothing \end{cases} \tag{2}$$

Свойства алгоритмов

1. **Отсутствие зависаний** *(starvation freedom / lockout freedom)*
   Каждый поток рано или поздно попадает в критическую секцию, т. е. каждое обращение к методам рано или поздно завершается.
2. **Неблокируемость** *(non-blocking)*
   Блокировка (задержка) одного потока не задерживает другие потоки.
3. **Свобода от взаимной блокировки** *(freedom from deadlock)*
   Хотя бы один из конкурирующих потоков должен рано или поздно попасть в нужную ему критическую секцию. Если некоторый поток не может войти в критическую секцию, то другие потоки должны завершить бесконечное количество критических секций.
4. **Честность** *(fairness)*
   Если D_A и D_B — попытки входа (взятия блокировки) в критические секции CS_A и CS_B для потоков A и B и $D_A \to D_B$, то $CS_A \to CS_B$.
5. **Свобода от ожидания** *(wait-free)*
   Выполнение метода заканчивается за конечное количество шагов без каких-либо процедур ожидания.
6. **Свобода от блокировок** *(lock-free)*
   Неограниченно частый вызов метода завершится за конечное число шагов.

Проблемы блокировок

- Крупнозернистая синхронизация плохо масштабируется
- Мелкозернистая тяжела в написании
- Тупики
- Инверсия приоритета
- Конвоирование

wait-free

- Гарантия безусловного прогресса для каждого потока в отдельности
- Гарантия безусловного прогресса системы в целом
- Обычно используются блокирующие память операции
- Реализации часто требуют $O(N)$ памяти, где N — число потоков

```
someClass::someClass() {
    ...
    __sync_fetch_and_add(referenceCount, 1);
}

someClass::~someClass() {
    ...
    if (__sync_sub_and_fetch(referenceCount, 1) == 0) {
        // delete unused shared object
    }
}
```

lock-free

- Уже нет гарантии безусловного прогресса для каждого потока в отдельности.
- Гарантия безусловного прогресса системы в целом, то есть хотя бы один поток продвигается вперёд независимо от внешних факторов
- Остальные потоки могут в это время активно ожидать, то есть, непродуктивно использовать процессорное время.
- Обычно используются CAS-примитивы.
- Может наблюдаться инверсия приоритетов и конвоирование.
- Контейнеры, как правило, быстрее wait-free.

```
void push(struct node *n) {
      do {
            struct node *t = top;
            n->next = t;
      while (!CAS(&top, t, n));
}
```

obstruction-free

- Условный прогресс: поток продвигается вперёд только тогда, когда нет конкуренции со стороны других потоков.
- Система в целом при большой взаимной конкуренции потоков не продвигается вперёд.
- Возможна живая блокировка (live-lock).
- Алгоритмы могут быть быстрее других.
- Некоторые алгоритмы можно реализовать лишь так (double-linked list).

- Свободный метод для изолированного потока обеспечивает завершение за конечное число шагов.
- Не гарантирует совместный прогресс.
- При наличии других потоков возможна взаимная блокировка.
- Все полны оптимизма, работают на себя и надеются на лучшее.