Concurrency 3

CS 2110 – Spring 2017

Announcements

- Course evaluations: we care.
- We care so much we make it **1% of your grade.**
- At the end of **May 11th,** we see a list of which students submitted evaluations.
- After grades are submitted, we see the anonymized content of the evaluations.
- We read them all.
- When giving feedback, please strive for **specificity** and **constructiveness.**

Axiomatic Basis for Computer Programming. Tony Hoare, 1969

Provide a definition of programming language statements not in terms of how they are executed but in terms of proving them correct.

{precondition P}
Statement S
{Postcondition Q}

Meaning: If P is true, then execution of S is guaranteed to terminate and with Q true

Assignment statement x= e;

{true} x= 5; {x = 5}	{x+1 >= 0} x= x + 1; {x >= 0}	{2*x = 82} x= 2*x; {x = 82}

Definition of notation:
P[x:= e] (read P with x replaced by e) stands for a copy of expression P in which each occurrence of x is replaced by e

Example: (x >= 0)[x:= x+1] = x+1 >= 0

Definition of the assignment statement:
{P[x:= e]}
x= e;
{P}

Assignment statement x:= e;

Definition of the assignment statement:
{P[x:= e]}
x= e;
{P}

{x+1 >= 0} x= x + 1; {x >= 0}	{2*x = 82} x= 2*x; {x = 82}

{2.0xy + z = (2.0xy + z)/6 }
x= 2.0*x*y + z;
{x = x/6}

x = x/6

2.0xy + z = (2.0xy + z)/6

5

If statement defined as an "inference rule":

Definition of if statement: If

{P && B} ST {Q} and
{P && !B} SF {Q}

Then
{P}
if (B) ST
else SF
{Q}

The then-part, ST, must end with Q true
The else-part, SF, must end with Q true

6

Hoare's contribution 1969:
Axiomatic basis: Definition of a language in terms of how to prove a program correct.

But it is difficult to prove a program correct after the fact. How do we develop a program and its proof hand-in-hand?

Dijkstra showed us how to do that in 1975.
His definition, called "weakest preconditions" is defined in such a way that it allows us to "calculate" a program and its proof of correctness hand-in-hand, with the proof idea leading the way.

Dijkstra: *A Discipline of Programming*. Prentice Hall, 1976.
A research monograph

Gries: The Science of Programming. Springer Verlag, 1981.
Undergraduate text.

7

How to prove concurrent programs correct.

Use the principle of non-interference

Thread T1	Thread T2
{P0}	{Q0}
S1;	Z1;
{P1}	{Q1}
S2;	Z2;
{P2}	{Q2}
…	…
Sn;	Zm;
{Pn}	{Qm}

T1 and T2 are proved correct in isolation.

What happens when T1 and T2 execute simultaneously?

How many execution orders are there?

8

How to prove concurrent programs correct.

Use the principle of non-interference

S1;	Z1;
S2;	Z2;
…	…
Sn;	Zm;

T1 and T2 are proved correct in isolation.

What happens when T1 and T2 execute simultaneously?

How many execution orders are there?

m+n instructions to execute:
- choose m of them for the Z's
- S's in the rest.

$$\binom{m+n}{m} = \frac{(m+n)!}{m! * n!} = a\ very\ big\ number$$

9

How to prove concurrent programs correct.

Thread T1	Thread T2
{P0}	{Q0}
S1;	Z1;
{P1}	{Q1}
S2;	Z2;
{P2}	{Q2}
…	…
Sn;	Zm;
{Pn}	{Qm}

Turn what previously seemed to be an exponential problem, looking at all executions, into a problem of size n*m.

Prove that execution of T1 does not interfere with the proof of T2, and vice versa.
Basic notion: Execution of Si does not falsify an assertion in T2: e.g. {Pi && Q1} S2 {Q1}

10

Interference freedom.

Susan Owicki's Cornell thesis, under Gries, in 1975.

Thread T1	Thread T2
{P0}	{Q0}
S1;	Z1;
{P1}	{Q1}
S2;	Z2;
{P2}	{Q2}
…	…
Sn;	Zm;
{Pn}	{Qm}

A lot of progress since then! But still, there are a lot of hard issues to solve in proving concurrent programs correct in a practical manner.

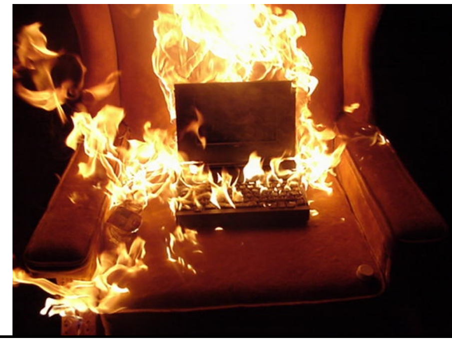
Prove that execution of T1 does not interfere with the proof of T2, and vice versa.
Basic notion: Execution of Si does not falsify an assertion in T2: e.g. {Pi && Q1} S2 {Q1}

11

The Harsh Truth

On the bright side…

A new way to melt your computer!



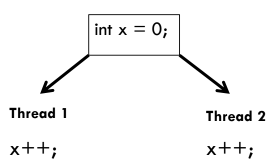
A new way to melt your computer!

```java
public class ForkBomb extends Thread {
    public static void main(String[] args) {
        (new ForkBomb()).start();
    }

    public @Override void run() {
        (new ForkBomb()).start();
        (new ForkBomb()).start();
    }
}
```

A new way to melt your computer!

Atomicity



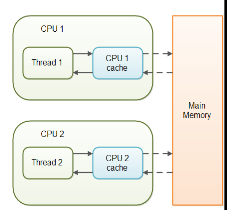
What is the value of x?

Can be either 1 or 2!

Caching and Volatile

- Concurrent programming is hard.
- Concurrent programming on real hardware is even harder!

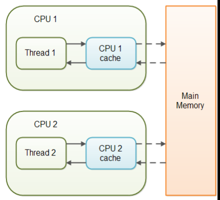
- Data is stored in caches
- Only written to main memory occasionally
- Huge efficiency gains!
- Huge concurrency headaches!



Caching and Volatile

- Concurrent programming is hard.
- Concurrent programming on real hardware is even harder!

- Volatile keyword
  - Fields can be declared **volatile**
  - All local changes are made visible to other threads
- **Does not guarantee atomicity!**
  - x+= 1 still does get, add, set; these may still be interleaved



Atomicity

volatile int x = 0;

Thread 1

x++;

Thread 2

x++;

What is the value of x?

Can be either 1 or 2!

Can we get atomicity without locks?

- class AtomicInteger, AtomicReference<T>, …
  - Represents a value
- method set(newValue)
  - has the effect of writing to a volatile variable
- method get()
  - returns the current value

- If the OS controls thread execution, how can the language ever guarantee atomicity?
  - New concurrency primitives: atomic operations.

Compare and Set (CAS)

- boolean compareAndSet(expectedValue, newValue)
  - If value doesn't equal expectedValue, return false
  - if equal, store newValue in value and return true
  - executes as a single atomic action!
  - supported by many processors – as **hardware instructions**
  - does not use locks!

```
AtomicInteger n = new AtomicInteger(5);
n.compareAndSet(3, 6); // return false – no change
n.compareAndSet(5, 7); // returns true – now is 7
```

Incrementing with CAS

```
/** Increment n by one. Other threads use n too. */
public static void increment(AtomicInteger n) {
        int i = n.get();
        while (n.compareAndSet(i, i+1))
                i = n.get();
}

// AtomicInteger has increment methods that do this
```

Lock-Free Data Structures

- Usable by many concurrent threads
- using only atomic actions – no locks!
- compare and swap is your best friend
- but it only atomically updates one variable at a time!

Let's look at one!

- Lock-free binary search tree [Ellen et al., 2010]
  http://www.cs.vu.nl//~tcs/cm/cds/ellen.pdf

Concurrency in other languages

- Concurrency is an OS-level concern
- Platform-independent languages often provide abstractions on top of these.
  - Java, Python, Matlab, …
- Different platforms have different concurrency APIs for compiled languages.
  - Unix/Linux: POSIX Threads (Pthreads)
  - Mac OS (based on Unix!): Pthreads, NSThread
  - Windows APIs
  - iOS: ??
  - Android: ??

Graph Search

- Do you need to **travel** to a node to visit it?

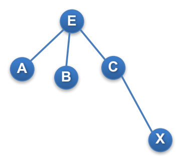
vs

Graph Search

- Do you need to **travel** to a node to visit it?
  - Depends on what information you have about the graph.
- Self-driving car (e.g., Uber) with nothing but sensors:
  - needs to explore to find its destination.
- Self-driving car (e.g. Waymo) with Google Maps:
  - compute a path, then follow it.

Graph Search

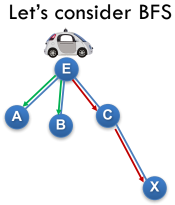
- Let's consider BFS.




```
/** Visit all nodes REACHABLE* from u.
Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Graph Search

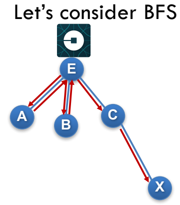
- Let's consider BFS if you're Google.




```
/** Visit all nodes REACHABLE* from u.
Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

Graph Search

- Let's consider BFS if you're Uber (no Google Maps!*).




```
/** Visit all nodes REACHABLE* from u.
Pre: u is unvisited. */
public static void bfs(int u) {
    Queue q= (u);
    while q is not empty) {
        u= q.popFirst();
        if (u has not been visited) {
            visit u;
            for each edge (u, v) leaving u:
                q.append(v);
        }
    }
}
```

*allegedly

If a method moves a robot…

- Your method's spec needs to say where the robot starts and ends in all possible scenarios.

```
/** Drive in a square with side length size, starting out
in the current direction. Car ends in the same location
and direction as it started. */
public void driveInSquare(int size) {
    for (int i = 0; i < 4; i += 1) {
            forward(size);
            turn(90);
    }
}
```

Wrapping up the course

- What is this course good for?
- Where can you go from here?

Coding Interviews

- A quick web search reveals: **We've taught you most of what you need for coding interviews.**
  - https://www.reddit.com/r/cscareerquestions/comments/20ahfq/heres_a_pretty_big_list_of_programming_interview/
  - http://maxnoy.com/interviews.html
  - …
- Your interviewer will be **impressed*** if you:
  - Write specs before you write methods.
  - Talk about/write invariants for your loops.
  - ...

*If not, don't work there.

What else is there?

- This course scratches the surface of many subfields of CS.
- Topics that have 4000-level courses:
  - Analysis of algorithms
  - Computational complexity
  - Compilers (parsing, grammars)
  - Programming Languages (formal semantics, ...)
  - Applied Logic (correctness proofs, ...)
  - Operating Systems (concurrency, caching, …)
  - Artificial Intelligence (graph searching, ...)
- ...among others.

*If not, don't work there.