Concurrency:
State Models & Design Patterns

Q&A Extended

Week 13

Exercise 12 - Task 2 - **Remaining Questions**

3 questions Safety / Liveness / Fairness

2 questions LTS/FSP

4 questions Petri Nets

6 questions Conceptual

1 question Organizational Affairs

Totally 16 Questions in 5 Categories

Exercise 12 - Task 2

Safety / Liveness / Fairness

Exercise 12 - Task 2 - Q039

Q:

Generally, how do we find out if something is fair without using the definitions?

A:

Fairness is a liveness property and thus hard to verify. So you really have to test every possible case in a model. This takes quite much time and is for larger objects under test not really feasible. In real life, often assumptions are made and simplifications performed to reduce the time requirements.

Exercise 12 - Task 2 - Q040

Q:

Liveness-Guarded Methods: When should you provide a policy for upgrading readers to writers? (I think a policy is implemented to give priority to writers so that the writers do not starve. The writers mostly starve when readers acquire locks unnecessarily.)

A:

You shouldn't upgrade them, because you would break the intention of the design (and thus increase excessively the complexity). You should tackle the problem of fairness at the lower levels instead. A possible starvation criteria could be the average thread waiting time, based on which the reader / writer ratio (or fairness implementation) could be adjusted dynamically during runtime.

Exercise 12 - Task 2 - Q041

Q:

Liveness and Guarded Methods, slide 50: Why do the first and second points ensure safety and the third point liveness?

A:

Ignore Interrupts: Preserves safety, not liveness, because empty catch clause = no error handling = unfinished operations that may lead to incomplete states (required info will never be acquired / fixed) = deadlock

Terminate: Preserves safety (no inconsistent state), but not liveness (stop)

Exit: Liveness preserved, because workflow is not entangled, but safety could be an issue, as others need to fix the encountered problem.

Exercise 12 - Task 2

LTS/FSP

Exercise 12 - Task 2 - Q042

Q:

I have some doubts about fsp modelling. I’m not sure if I can be able to model so complex processes. Should we expect complex fsp modelling questions in the exam? How much information do we need to know for fsp modelling? I mean, should we able to model “priority” with fsp?

A:

You don't need to know all about fancy LTS / FSP features. However, you should be able to understand and adapt the LTS models in the slides and especially the exercises. That doesn't mean that you must be able to build the whole implementation from scratch, but you should be able to implement the key parts. That's often not more than 3 - 4 lines of code.

Exercise 12 - Task 2 - Q043

Q:

Series 03, Ex 4: Why in S1 you do not add something link (like?) pop[u: u != v] -> error -> S1 and the analogous for S2?

A:

Two reasons:

1) In pop[...] of S1 you don't have access to u and v. The term enclosed by the square brackets is the current stack content. The values get forwarded to S2 (stack has two items) or S1 (stack has one item) or S0 (empty stack). From another point of view: It has to be the same value, because otherwise the action cannot execute at all. You would have to add another action from S0 calling with wrong arguments, but then we just get to 2) :)

2) Such (manual) checks would introduce VERY MUCH additional overhead (additional states, ...).

Exercise 12 - Task 2

Petri Nets

Exercise 12 - Task 2 - Q044

Q:

CP-11, slide 76: What is "zero-testing" in general? (And what are zero-testing nets?)

A:

Zero-testing is highly dependent on the context (x==0, place is empty in Petri net, ...) and can mean completely different things.

Zero-testing in the context of Petri-nets introduces a new type of places (zero-input places), which let transitions only become active when they don't contain any tokens. Consequently, zero-testing nets are nets that contain at least one zero-input place (see example in slides).

The important thing here is that ordinary Petri nets have no ability to perform arbitrary operations on numbers of tokens to be expressed. You will find more (quite complex) info about Turing-complete extensions at

http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6463459

Exercise 12 - Task 2 - Q045

Q:

What is a feature in the context of a Petri net?

A:

- Place

- Transition

- Input Function

- Output Function

- Marking

Exercise 12 - Task 2 - Q046

Q:

CP-11, slide 79: FSA - Petri net - Turing-completeness (in the context of Petri nets)?

A:

See answer for Q044.

Exercise 12 - Task 2 - Q047

Q:

Where in the Chomsky hierarchy are Petri nets?

A:

Between Type-2 (Context-free) and Type-3 (Regular). It was shown that all regular languages are Petri net languages and the family of Petri net languages are strictly included in the family of context-sensitive languages but some Petri net languages are not context-free and some context-free languages are not Petri net languages. It was also shown that the complement of a free Petri net language is context-free.

more info in at

https://www.intechopen.com/books/petri-nets-manufacturing-and-computer-science/grammars-controlled-by-petri-nets

Exercise 12 - Task 2

Conceptual

Exercise 12 - Task 2 - Q048

Q:

What are the difficulties of creating a bounded thread pool?

A:

1) The choice of the appropriate bound parameters. Too little or too many concurrent threads delay the execution.

2) You have to assign/split the work in a way it fits for a specific amount of threads.

Exercise 12 - Task 2 - Q049

Q:

Briefly explain one distributed lock algorithm of your choosing (for example: RedLock) and its limitations.

A:

Redlock is a client side distributed locking algorithm designed to be used with Redis, but the algorithm orchestrates, client side, a set of nodes that implement a data store with certain capabilities, in order to create a multi-master fault tolerant, and hopefully safe, distributed lock with auto release capabilities.

Limitations:

It just prevents a (more or less) synchronized interface, but you can still introduce deadlocks (logic errors) in your own code.

Exercise 12 - Task 2 - Q050

Q:

In the dining philosophers scenario, are there models that deal with the occurrence of processes, that have crashed and therefore unable to release a resource?

A:

Dealing with such processes would involve timeouts or liveness checks (polling, ...). You can find various models and adapt them to your personal needs from the large-scale distributed systems field.

Exercise 12 - Task 2 - Q051

Q:

Could you tell about best practices in concurrent programming?

A:

- Simplify your model as much as possible (less is more)

- Keep your synchronized blocks as short and rare as possible

- For larger projects: use already available synch solutions (embedded in data bases, ...)

Exercise 12 - Task 2 - Q052

Q:

What are actual trends in concurrent programming?

A:

- GPU computation (fast for matrix operations)

- Heterogeneous CPUs (ARM big.little, co-processors)

These require new/other development tools.

Exercise 12 - Task 2 - Q053

Q:

Why (and how) would you use inner classes to implement asynchrony?

A:

- Obfuscation?

- Bad practice in my opinion

```
public class Temp2 {

    public static void main(String[] args) {
        Temp2 temp2 = new Temp2();
        InnerClass ic = temp2.new InnerClass();
        ic.start();
    }

    public class InnerClass extends Thread {
          public synchronized void methodA(){
              System.out.println("Hi there!");
          }

          public void start() {
              this.methodA();
          }
    }
}
```

You can instantiate the inner class and create a thread out of it.

Exercise 12 - Task 2

Organizational Affairs

Exercise 12 - Task 2 - Q054

Q:

I won’t attend the lecture on Dec 13th, because of a seminar meeting. Can you share your answers as pdf? Also, Can you share the final exam of the previous lecture?

A:

1) Yes, already done.

2) Done as well. :)


Good Luck!