Concurrency:
State Models & Design Patterns

Practical Session

Week 04

CP: Concurrency: State Models and Design Patterns

Course: 21005 (Autumn Semester 2019)
Lecturer: Prof. Oscar Nierstrasz
Assistants: Pascal Gadient, Mohammadreza Hazhirpasand
Lecture: Wednesday, 10h15 - 12h00
Exercises: Wednesday, 12h00 - 12h45
Place: Engehaldenstrasse 8, 003
Start: 2019-09-18
Exam: 2019-12-18
Repetition: Autumn 2021

Description

This course provides an introduction to concurrent programming with Java. The course focuses on *fundamental concepts* important for developing correctly functioning concurrent programs, such as safety, liveness and fairness, and on *standard programming patterns* and *techniques* for dealing with these issues. The course will include two lab sessions (replacing the regular lecture hours) in which students will work in small groups to apply the techniques presented.

Much of the practical material in this lecture will be based on: Doug Lea, *Concurrent Programming in Java: Design Principles and Patterns*, Addison-Wesley, 1999. The theoretical foundations will be based on: Jeff Magee and Jeffrey Kramer, *Concurrency: State Models & Java Programs*, John Wiley, 1999.

Learning Outcomes

On successful completion of this course, you will be able to:

- Reason about safety, liveness and fairness in concurrent programs
- Use model-checking tools to prove safety and liveness properties
- Use synchronization mechanisms to guarantee thread safety in programs
- Reason about communication mechanisms to manage threads
- Use practical techniques to avoid deadlock and ensure liveness
- Reason about architectural styles to avoid concurrency issues

Grading

The final grade in the course will be based 30% on exercises and 70% on the final exam.

Course Schedule

1	18-Sep-19	Introduction
2	25-Sep-19	Java and Concurrency
3	2-Oct-19	Safety and Synchronization
4	9-Oct-19	Safety Patterns + Transactional Memory
5	16-Oct-19	Liveness and Guarded Methods
6	23-Oct-19	Lab session
7	30-Oct-19	Liveness and Asynchrony

MSc registration Fall 2019

JMCS students

- **Register on Academia for teaching units by Oct. 11, 2019**
- **Register on Academia** for the **CP exam** by **December 4, 2019**
  - *[January 3, 2020 for all other MSc courses]*

NB: Hosted JMCS students *(e.g. CS bachelor students etc.)* must additionally:

- **Request for Academia access** by **September 30, 2019**

See also: Student's ToDo list for every semester

Registration

- Please also register on Piazza.com

Resources

- Annotated lecture slides (pdf)
- Exercises (pdf)
- LTSA and Java examples repo
- LTSA 3.0 download
- LTSA documentation
- Petit Petri Net Interpreter
- Course evaluations: WS05/06, HS10, HS15, HS17

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

Reading material

- Concurrent programming in Java (unibe access only)
- Online Supplement to *Concurrent Programming in Java* by Doug Lea.
- java.util.concurrent Interest Site
- Concepts and Notations for Concurrent Programming survey paper (pdf)
- Petri Nets survey paper (pdf)
- Linda survey paper (pdf)

You should be able to access them through:
- Uni BE VPN
- “eduroam” WLAN

If you still cannot access the PDF resources, please send me an email.

Assignment 03

Discussion

A03 - Exercise 1

a) What are safety properties? How are they modeled in FSP?

A safety property P consists of (textual or mathematical) definitions describing properties of a model. These definitions can be implemented within a deterministic process.

b) Is the busy-wait mutex protocol fair? Deadlock-free? Justify your answer.

This protocol is fair because a process always gives priority to the other process to enter the critical section and it has been formally proven to be deadlock free (see the FSP in the lecture).

A03 - Exercise 1

c) Can you ensure safety in concurrent programs without using locks?

No, to ensure safety you need some sort of locks, because you have to guarantee that only one thread at a time can access shared resources.

d) The Java language designers decided to implement concurrency based on monitors. What is the main reason behind this decision? What other options except monitors could have been chosen?

The notion of monitors is similar to the notion of objects. The concepts of semaphores and message passing could also be used for ensuring safety and liveness, but because of their complex configurations these are not that easily adaptable to existing language specifications.

A03 - Exercise 2

Consider the following process definitions. Are T1 and T2 equivalent? Why?

```
R = (a->c->R).
S = (b->c->S).
||T1 = (R || S).
T2 = (a->b->c->T2|b->a->c->T2).
```

Yes. The only possible action traces are: “a, b, c” or “b, a, c” for both processes.

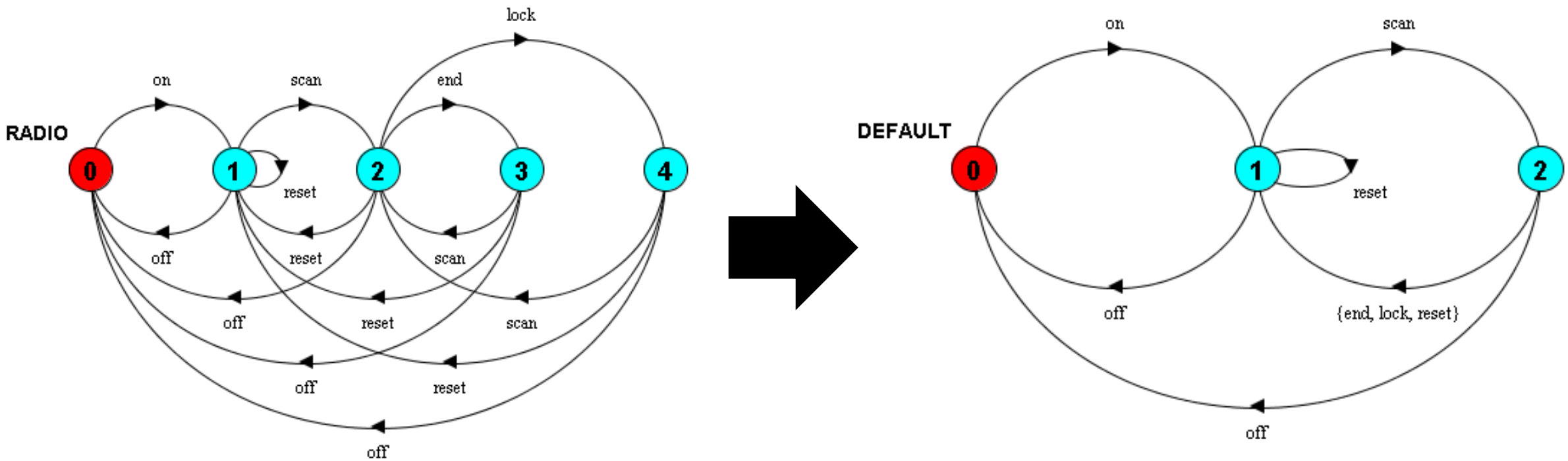
A03 - Exercise 3

A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons, scan and reset, which operate as follows. (...)

```
RADIO = OFF,
OFF = (on-> TOP),
BOTTOM = (off -> OFF | scan -> SCANNING | reset -> TOP),
TOP = (off -> OFF | scan -> SCANNING | reset -> TOP),
SCANNING = (off -> OFF | lock -> LOCKED | end -> BOTTOM | reset -> TOP),
LOCKED = (off -> OFF | scan -> SCANNING | reset -> TOP).
```

A03 - Exercise 3

A miniature portable FM radio has three controls. An on/off switch turns the device on and off. Tuning is controlled by two buttons, scan and reset, which operate as follows. (...)



A03 - Exercise 4

Consider the following problem specification:

a) We have a stack that has two slots.

b) The stack operations pop and push should be atomic. In other words, the stack is locked while being popped or pushed.

c) The input space is 0..1 (You can only push 0s and 1s).

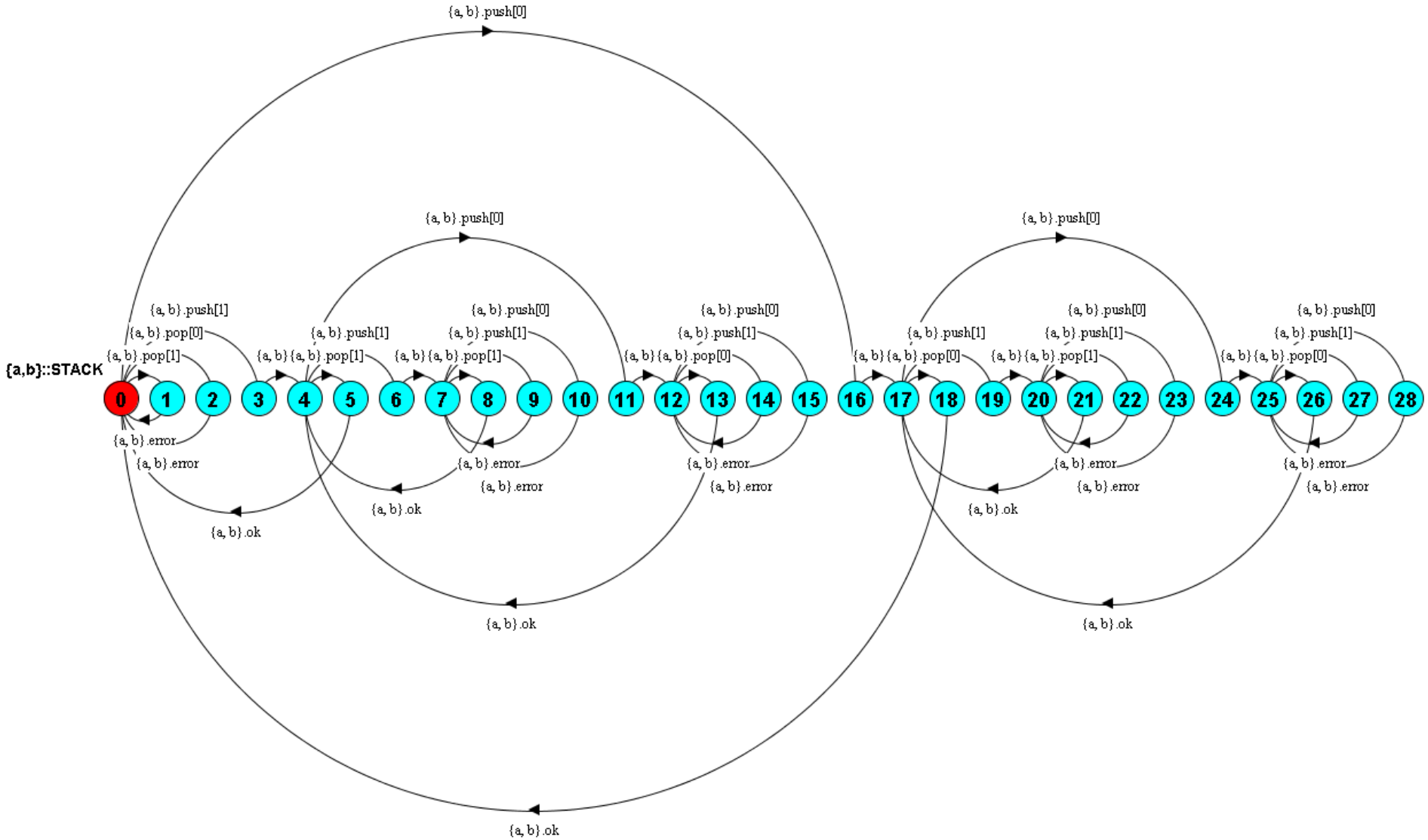
d) There is one producer process that pushes 0s and 1s repeatedly.

e) There is one consumer that pops the top of the stack.

f) Throw an error for pushing to a full stack or popping from an empty stack.

Write an FSP description for this problem and verify your model using LTSA.

A03 - Exercise 4



Assignment 04

Preview

A04 - Exercise 1

Please answer the following questions:

a) Why are immutable classes inherently safe?

b) What is “balking”?

c) When is partial synchronization better than full synchronization?

d) How does containment avoid the need for synchronization?

e) What liveness problems can full synchronization introduce?

f) When is it legitimate to declare only some methods as synchronized?

A04 - Exercise 2

The dining savages: A tribe of savages eats communal dinners from a large pot that can hold M servings of stewed missionary. When a savage wants to eat, he helps himself from the pot unless it is empty in which case he waits for the pot to be filled. If the pot is empty the cook refills the pot with M servings.

The behavior of the savages and the cook are described by:

```
SAVAGE = (getserving -> SAVAGE).
COOK = (fillpot -> COOK).
```

Model the behavior of the pot as an FSP process.

A04 - Exercise 3

Please consider the FSP below and answer the questions:

```
property LIFTCAPACITY = LIFT[0],
      LIFT[i:0..8] = (enter -> LIFT[i+1]
      |when(i>0) exit -> LIFT[i-1]
      |when(i==0)exit -> LIFT[0]).
```

a) Which values can the variable I have in non-error states?

b) What kind of property is used and what does it guarantee?

c) Provide an action trace that violates the provided property.

d) Provide an action trace that does not violate the provided property.

A04 - Exercise 4

Implement a thread-safe MessageQueue class in Java using one of the safety patterns. You have to justify your choice of the safety pattern.

- The MessageQueue has the following specifications:
- It provides **FIFO** (first-in-first-out) access
- (...)
- There are (at least) **two methods**: `add(String msg)` and `remove()`
- **Write a unit test for your MessageQueue to demonstrate the thread-safety of your implementation**

*You have to attend the lecture to reveal such slides.**

:-)

**Disclaimer:*

The content that has been shown on this slide is irrelevant for the exam.