Concurrency:
State Models & Design Patterns

Practical Session

Week 04

Exercises 03

Discussion

Exercise 03 - Task 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).

Exercise 03 - Task 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.

Exercise 03 - Task 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.

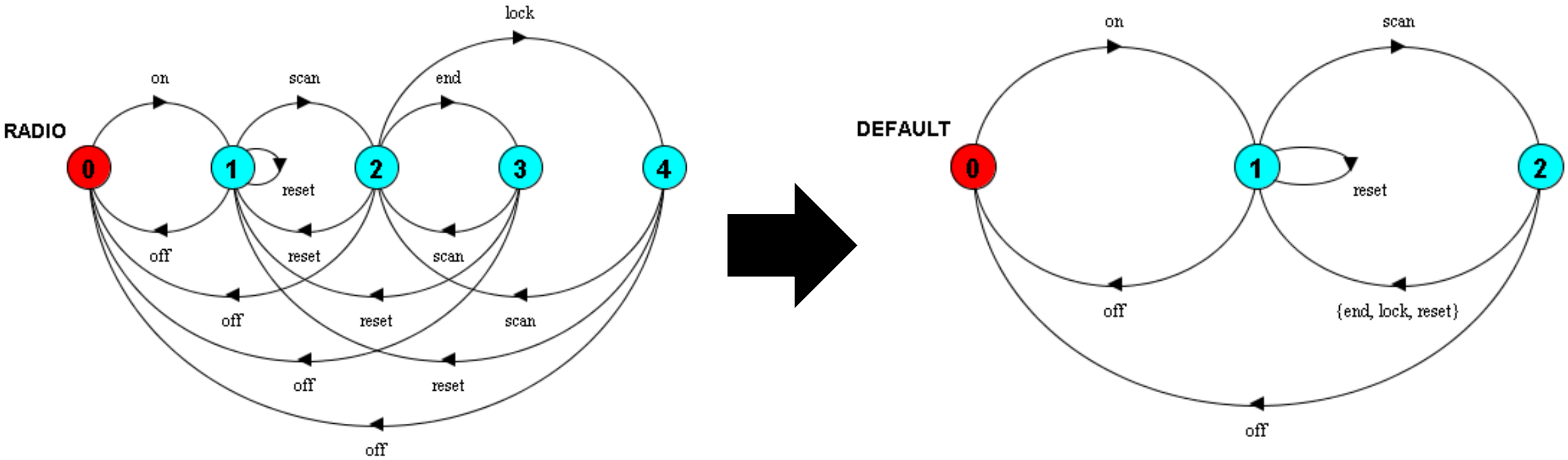
Exercise 03 - Task 3

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).
```

Exercise 03 - Task 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. (...)



Exercise 03 - Task 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).

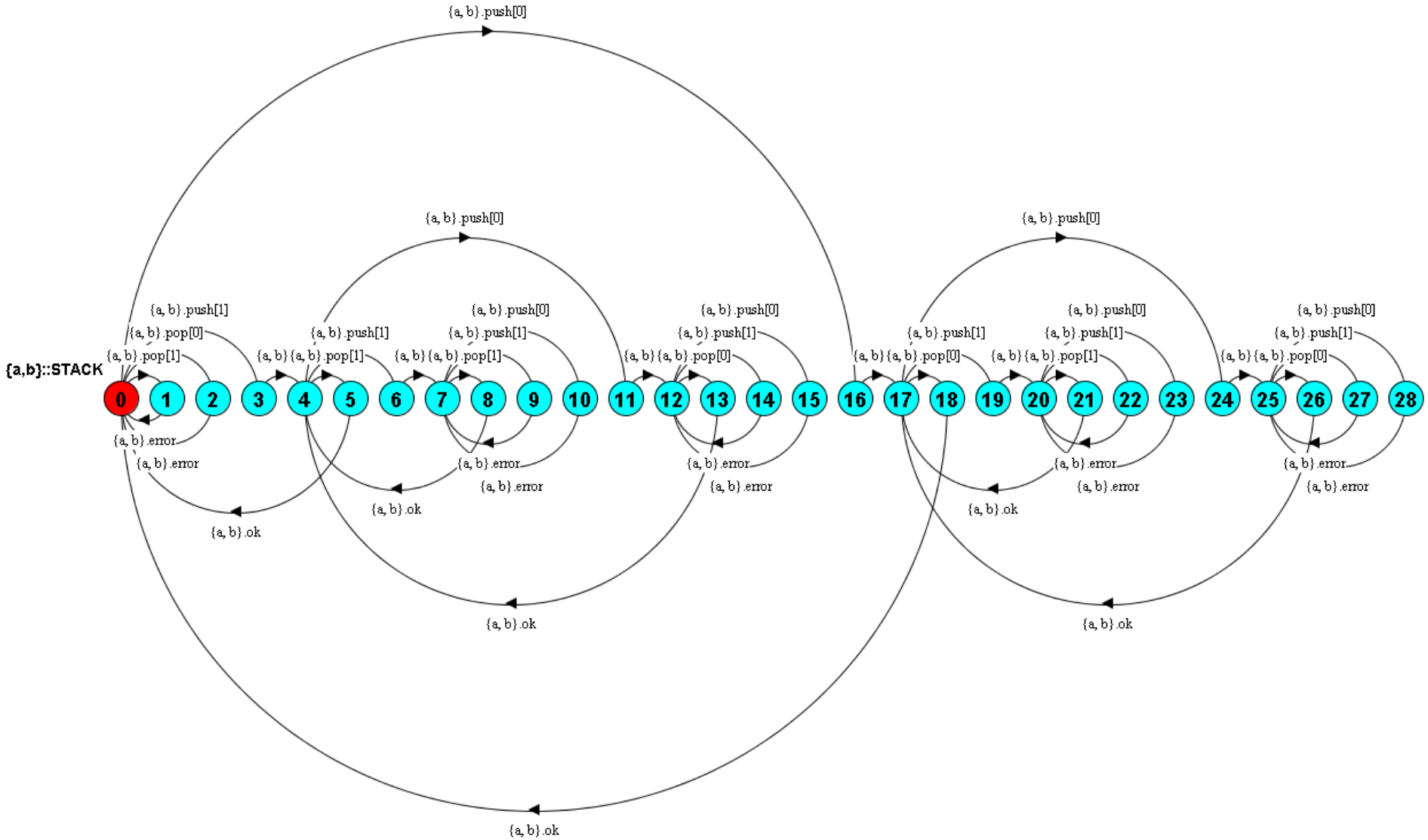
d) There is one producer process that pushes 0s and 1s repeatedly.

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.

Exercise 03 - Task 4



Exercises 04

Preview

Exercise 04 - Task 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?

Exercise 04 - Task 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.

Exercise 04 - Task 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 take?

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.

Exercise 04 - Task 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**