Logics and Processes Algebra

Second year master's degree in
artificial intelligence
Prof. Mustapha Bourahla

Exercises (Series 1)

Exercise: Using the inference rules draw LTS of :

1. process one [a,b,c] a; (b; stop [] c; stop) endproc

2. process two [a,b,c] a; b; stop [] a; c; stop endproc

3. process3 := a; (b; d; stop [] c; stop)

4. process4 := a; b; d; stop [] a; c; stop

Solutions:

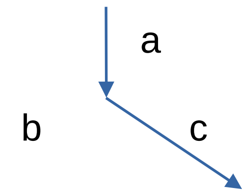
En utilisant les règles d’inference dessiner les arbres (STE) de

process one [a,b,c] a; (b; stop [] c; stop) endproc

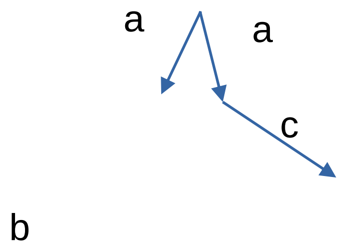
a; ((b; stop [] c; stop)) --- a > (b; stop [] c; stop)

(b; stop [] c; stop) --- b > stop

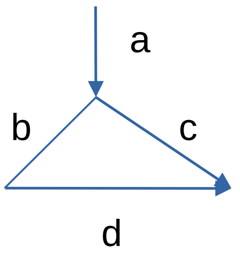
(b; stop [] c; stop) --- c > stop



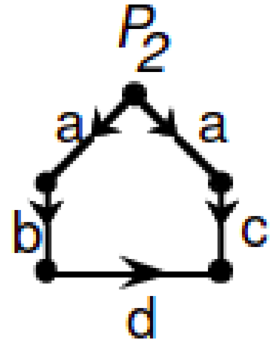
process two [a,b,c] a; b; stop [] a; c; stop endproc



P1 := a; (b; d; stop [] c; stop)



P2 := a; b; d; stop [] a; c; stop



Exercises (Series 2)

Exercise 1: Give the LTS of: a; (b; stop [] c; stop) and a; b; stop [] a; c; stop. Then give a conclusion

Exercise 2: Give the LTS of each: A:= mon; (gom;stop [] choc; stop), B := mon; gom ;stop [] mon; choc; stop, C := mon; (i; gom; stop [] mon; choc; stop), and D := mon; (i; gom; stop [] i; mon; choc; stop)

Exercise 3: Marie and Abdel always eat together. They have three actions: Breakfast (b), lunch (l), dinner(d):
Marie:= b; l; d; stop, Abdel:= b; l; d; stop, give the LTS of Marie || Abdel

Exercise 4: However, if Abdel is not used to having lunch:

Marie:= b; l; d; stop, Abdel:= b; d; stop, give the LTS of Marie || Abdel

Exercises (Series 3):

Exercise 1: prove the following equivalences:

- ((a; stop || a; stop) || a; stop) = a; stop
- ((hide a in (a; stop || a; stop)) || a; stop) = i; stop
- (hide a in ((a; stop || a; stop) || a; stop)) = i; stop

Exercice 2: Marie and Abdel have nothing to do with each other. They have two actions: Breakfast (b), lunch (l): Marie:= b; l; stop, Abdel:= b; l; stop. find Marie ||| Abdel

Exercise 3: Marie and Abdel make breakfast and dinner separately, however they always eat lunch together : Marie:= b; l; d; stop, Abdel:= b; l; d; stop. Give Marie |[l]| Abdel

Exercise 4: compute (a; b; stop [] c; d; stop) |[a,b]| (a; b; stop [] d; f; stop) and give its LTS

Exercise 5: compute a; b; c; stop |[b]| a; b; d; stop

Exercises (Series 4)

Exercise 1: verify

1. (a; b; stop) |[b]| (c; b; stop) = (a; c; b; stop) [] (c; a; b; stop)
2. (i; b; stop) |[b]| (c; b; stop) = (i; c; b; stop) [] (c; i; b; stop)
3. (i; b; stop) |[b]| (i; b; stop) = (i; i; b; stop) [] (i; i; b; stop) = (i; i; b; stop)
4. (a; b; stop) |[b]| (b; c; stop) = a; b; c; stop
5. (a; b; stop) |[a, b]| (b; a; stop) = stop = (a; b; stop) || (b; a; stop)
6. (a; b; stop [] d; f; stop) |[a, b]| (a; b; c; stop [] i; stop) = (a; b; c; stop [] d; (f; i; stop [] i; f; stop) [] i; d; f; stop)

Exercises (Series 5)

Exercise 1: Hello World!

Consider a system without subsystem and that performs a single actions: saying "Hello World". Give the code for this scenario and represent it graphically. Execute it.

Exercise 2: Greatest Common Divisor

Design a Scola model that calculates the greatest common divisor (GCD) of two integers. Execute it with a=96 and b=81.

Hint: recall that GCD(a, a) = a and that GCD(a, b) = GCD(a, b-a) if a<b.

Exercise 3: Syracuse Problem (Collatz conjecture)

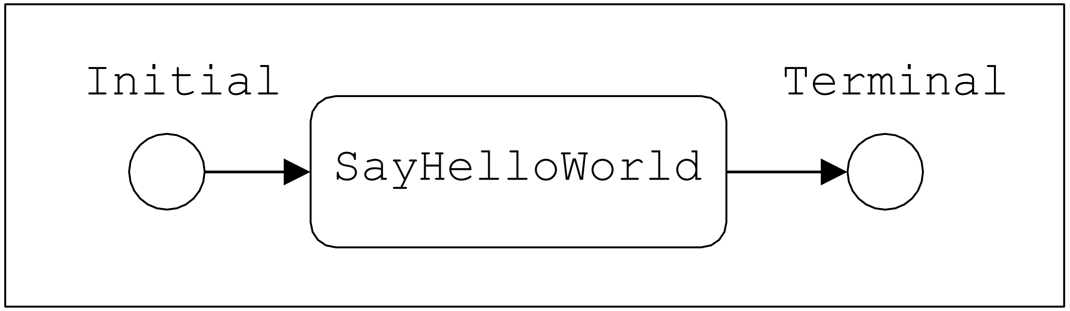
Design a Scola model that takes any integer n and performs the following operations:

- If n is equal to 1, the execution stops.
- If n is even (n modulo 2 = 0), then the execution goes on with n/2.
- If n is odd (n modulo 2 = 1), then the execution goes on with 3n+1.

Execute this model for n=19.

Scola operations for multiplication and the modulo are respectively `mul` and `mod`.

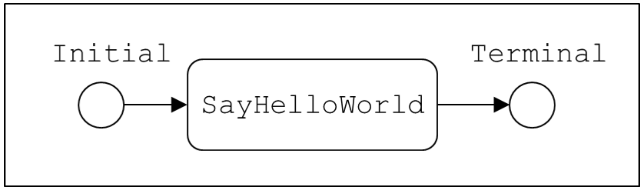
Hello World! (1)



The system is made of a single (implicit) block.
The scenario consists in starting from an initial state, going through a task SayHelloWorld and ending up into a terminal state.

Hello World! (2)

```
scenario HelloWorld
    state Initial
    task SayHelloWorld end  state
    Terminal
    next Initial SayHelloWorld
    next SayHelloWorld Terminal
end
```

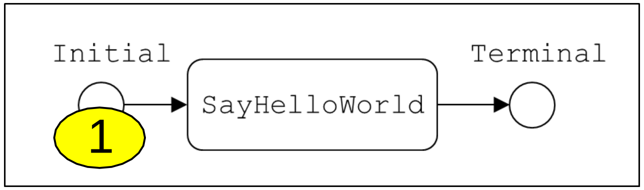


- Tasks are container for instructions. This is the reason why their declarations end with the keyword "end".
- Next instructions represent arrows. They are used to chain states, tasks and gateways.
- The order of declaration of elements is irrelevant:

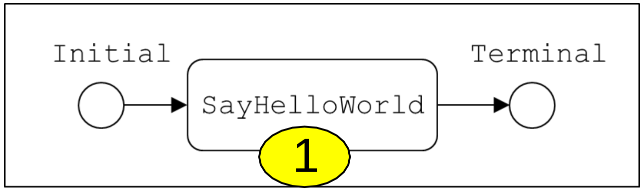
```
scenario HelloWorld
    state Initial
    next Initial SayHelloWorld  task
    SayHelloWorld end  next
    SayHelloWorld Terminal  state
    Terminal
end
```

Hello World! (3)

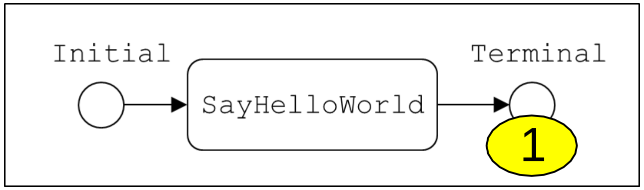
Scola models describe the evolution of processes. A **process** has a number, possibly a **parent process** and a number of **child processes**. Processes can be dynamically created by the execution of a model. At any step of the execution, a process is either inactive, or it is located on a state, a task or a gateway.



Step 1: a process number 1 is created on the initial state.

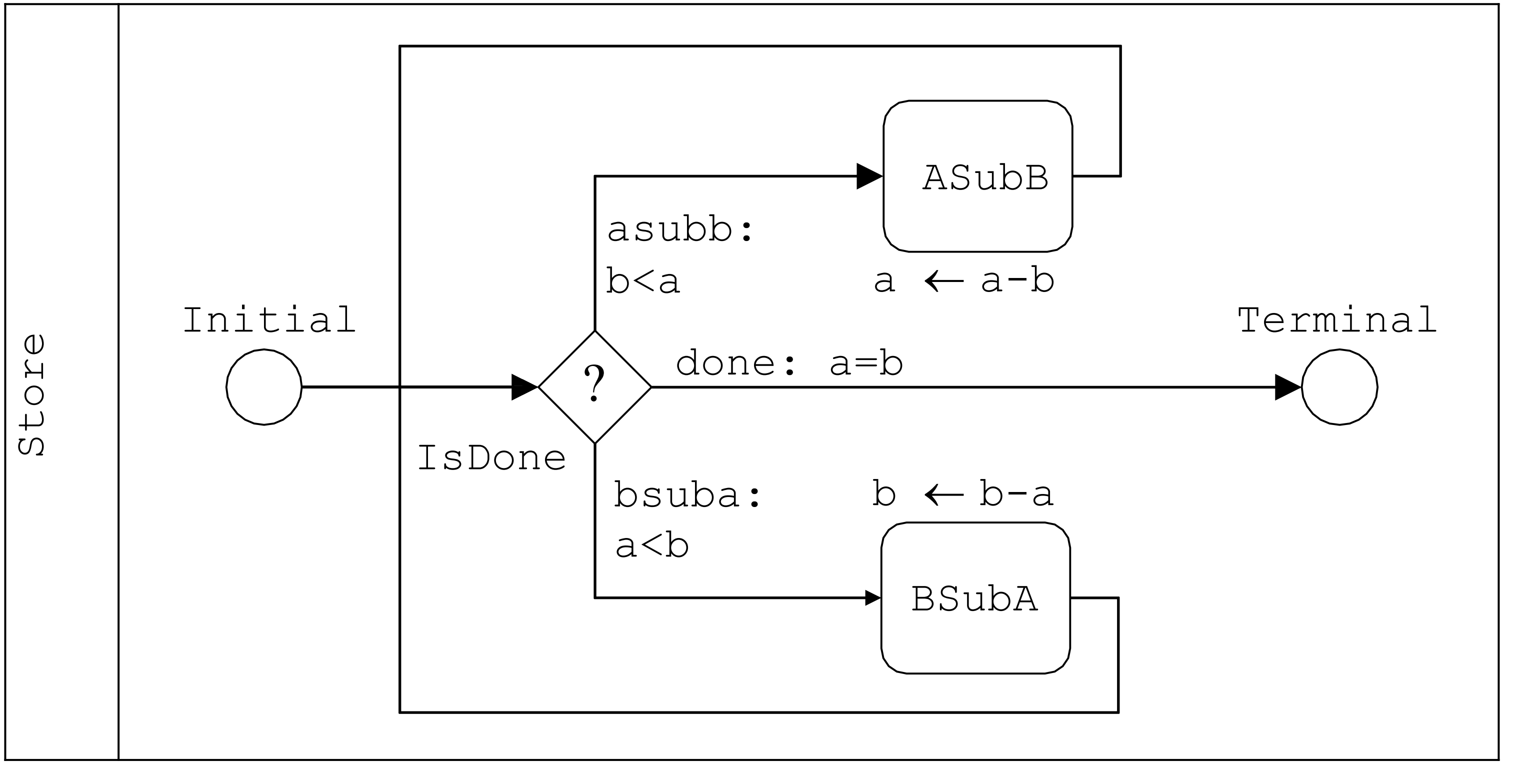


Step 2: the process moves to the task SayHelloWorld and performs this task.



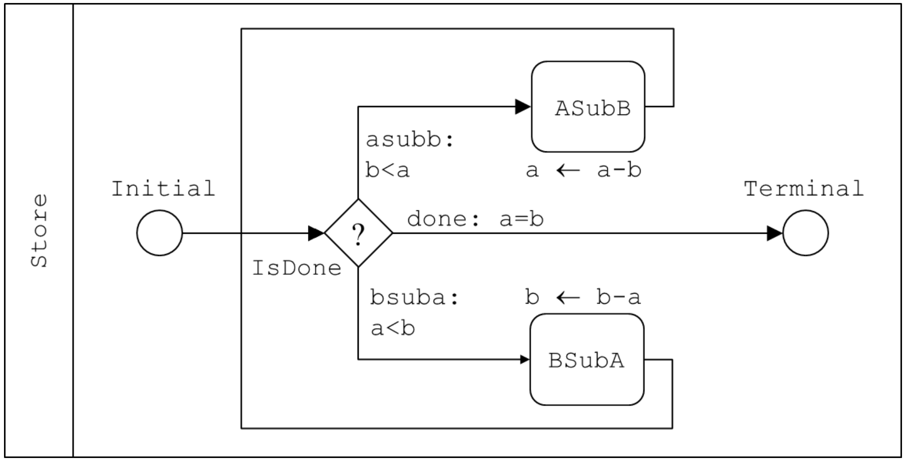
Step 3: the process moves to the terminal state. Its execution is finished.

Greatest Common Divisor (1)



Greatest Common Divisor (2)

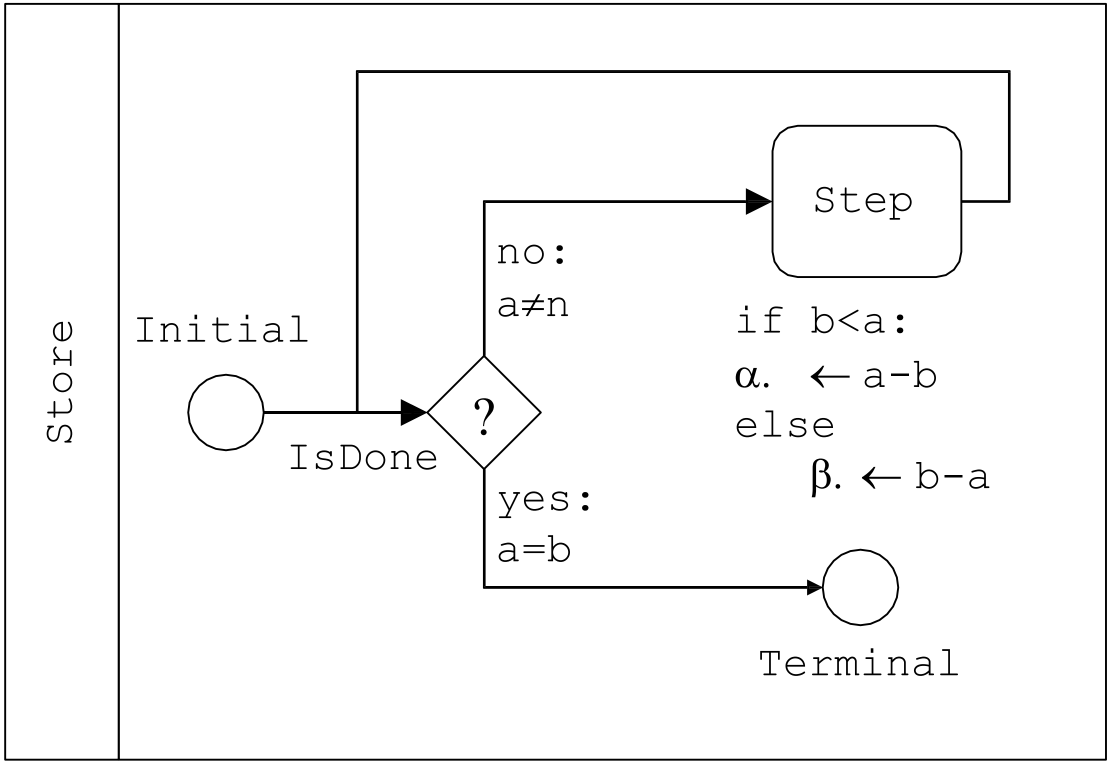
```
scenario GCD as Store
   state Initial
   test IsDone
      case done (eq a b)
      case asubb (lt b a)
      case bsuba (lt a b)
   end
   task ASubB
      set a (sub a b)
   end
   task BSubA
      set b (sub b a)
   end
   state Terminal
   next Initial IsDone
   next Reset Increment
   next IsDone.done Terminal
   next IsDone.asubb ASubB
   next IsDone.bsuba BSubA
   next ASubB IsDone
   next BSubA IsDone
end
```



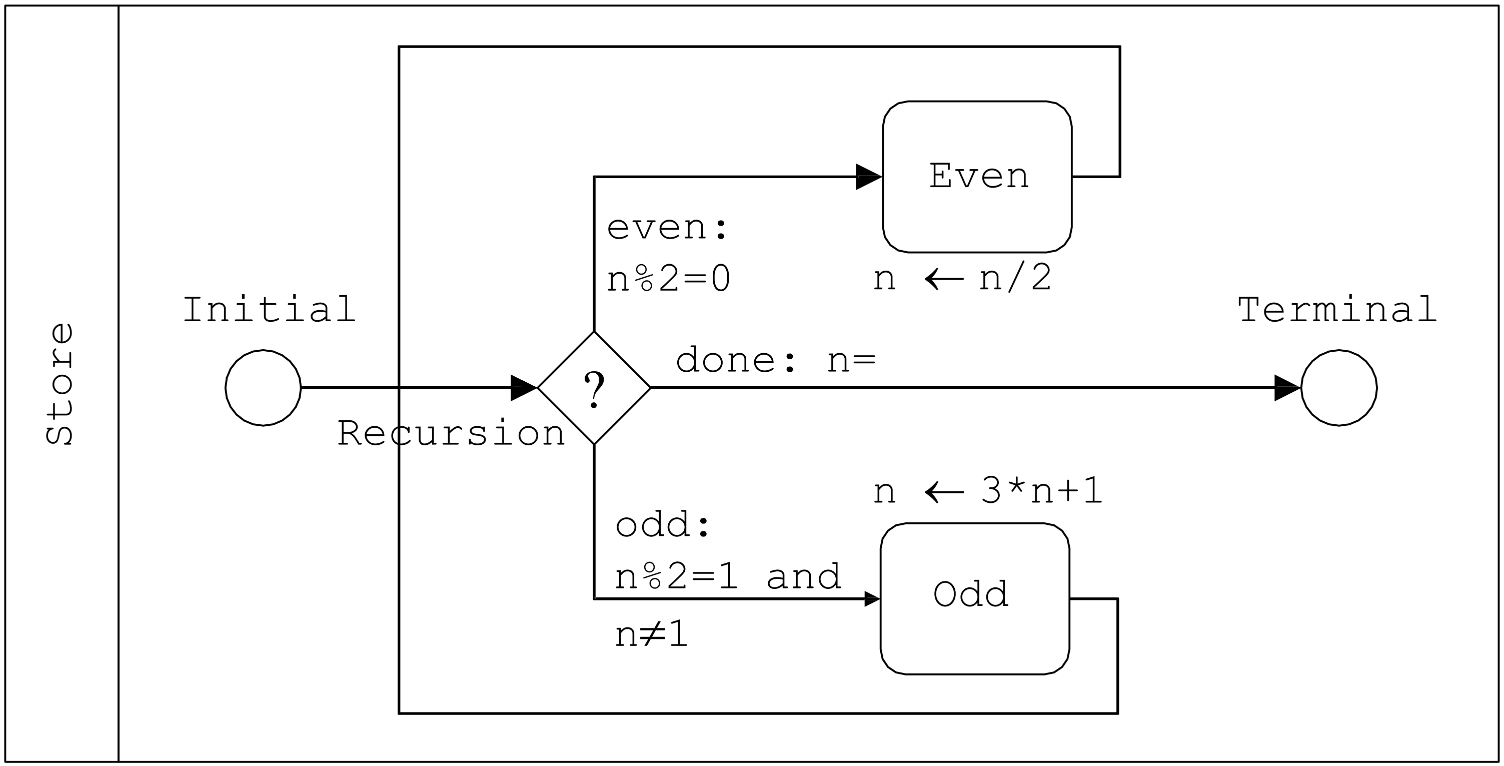
Greatest Common Divisor (3)

```
block Store
   integer a 96
   integer b 81
end

scenario GCD as Store
   state Initial
   test IsDone
      case yes (eq a b)
      case no (df a b)
   end
   task Step
      if (lt a b) then
         set b (sub b a)
      else
         set a (sub a b)
   end
   state Terminal
   next Initial IsDone
   next IsDone.yes Terminal
   next IsDone.no Step
   next Step IsDone
end
```

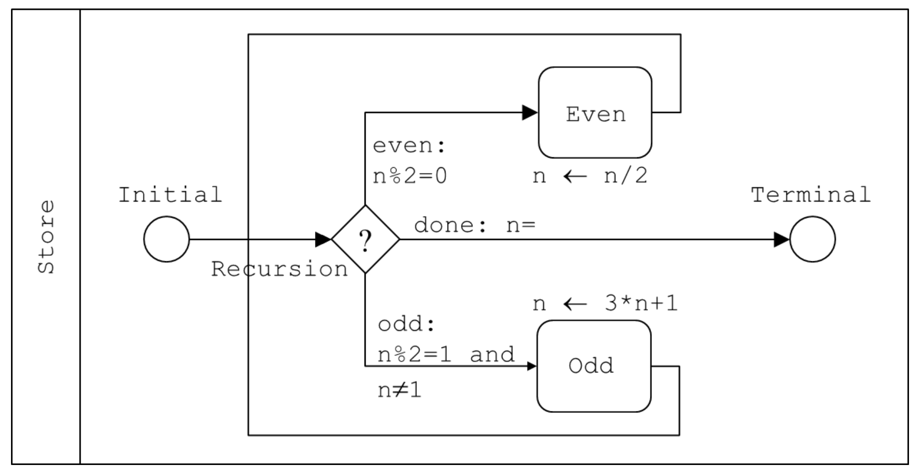


Syracuse (1)


Store
Even
even:
n%2=0
n ← n/2
Initial
Terminal
done: n=
?
Recursion
n ← 3*n+1
odd:
n%2=1 and
n≠1
Odd

Syracuse (2)

```
scenario Syracuse as Store
   state Initial
   test Recursion
      case done (eq n 1)
      case even (eq (mod n 2) 0)
      case odd (and (eq (mod n 2) 1) (df n 1))
   end
   task Even
      set n (div n 2)
   end
   task Odd
      set n (add (mul n 3) 1)
   end
   state Terminal
   next Initial Recursion
   next Recursion.done Terminal
   next Recursion.even Even
   next Recursion.odd Odd
   next Even Recursion
   next Odd Recursion
end
```



Whether this series converges to 1 for all value of n is still an open question.

Exercises (Series 6)

Exercise 1: At the restaurant.

At the restaurant, the client orders a pizza to the waiter. The waiter transmit the order to the cook, who bakes the pizza. Once the pizza is baked, the cook gives it to the waiter, who brings it to the client. Eventually, the client eats the pizza.

Represent and execute this scenario.

Exercise 2: Car assembly

In a car assembly line, the first station paints the car's body, the second assemble the engine and the third the wheels.

Represent and execute this scenario.

Exercises (Series 7)

Exercise 1: Life-Cycle.

The life-cycle of a product is usually made of three phases: design, operation and decommissioning. The operation phase is itself decomposed into two sub-phases: production and maintenance.

Give the code that represent such a life-cycle and represent it graphically. Execute it.

Exercise 2: Ternary Meter

Design a Scola model to represent a meter with three wheels (like a kilometric meter) that counts in base 3.

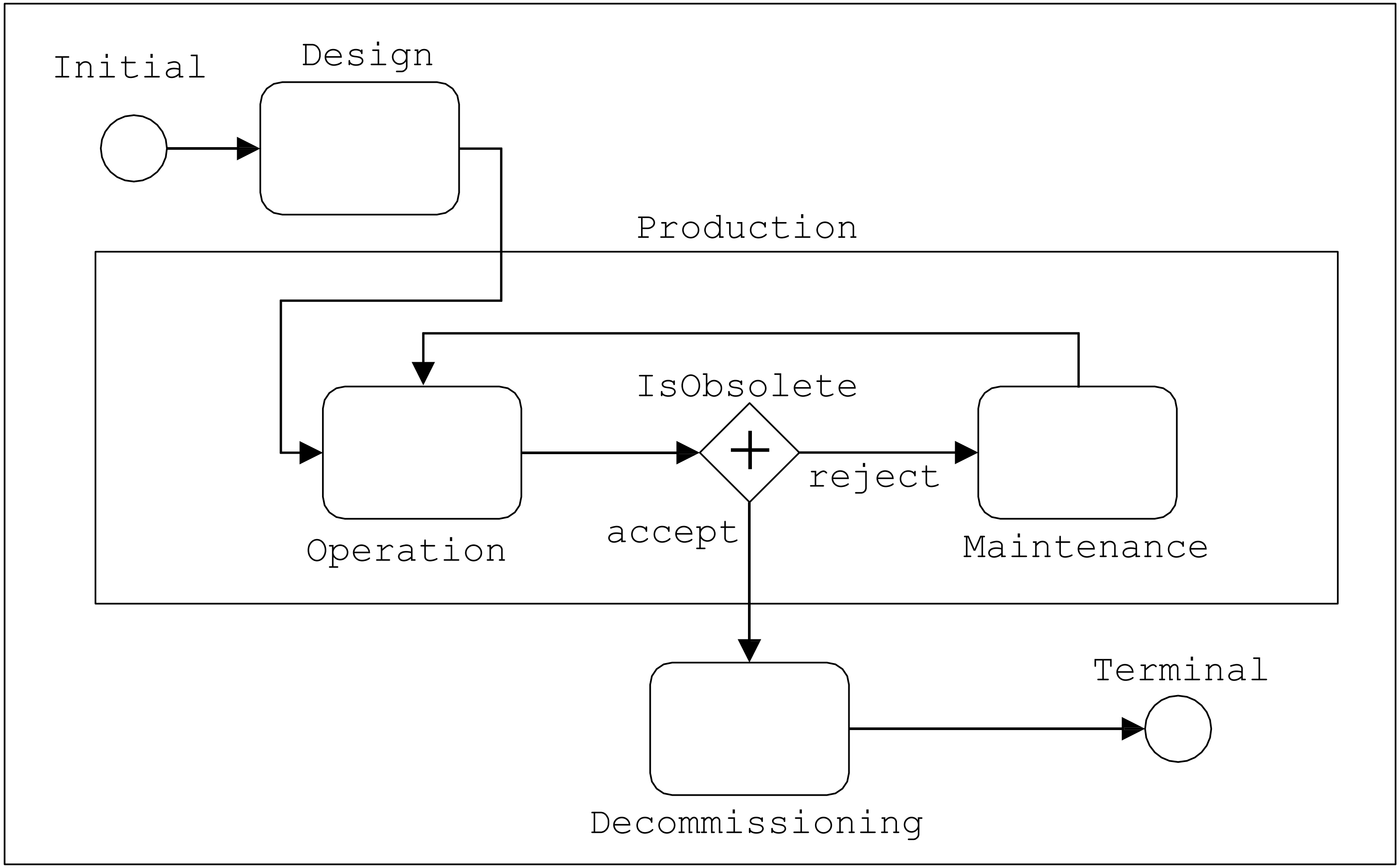
Exercise 3: Tapes and Siphons

Design a Scola model that, at the one end, creates as many processes as the analyst wishes (a tape) and, at the other end, kills these processes (a siphon).

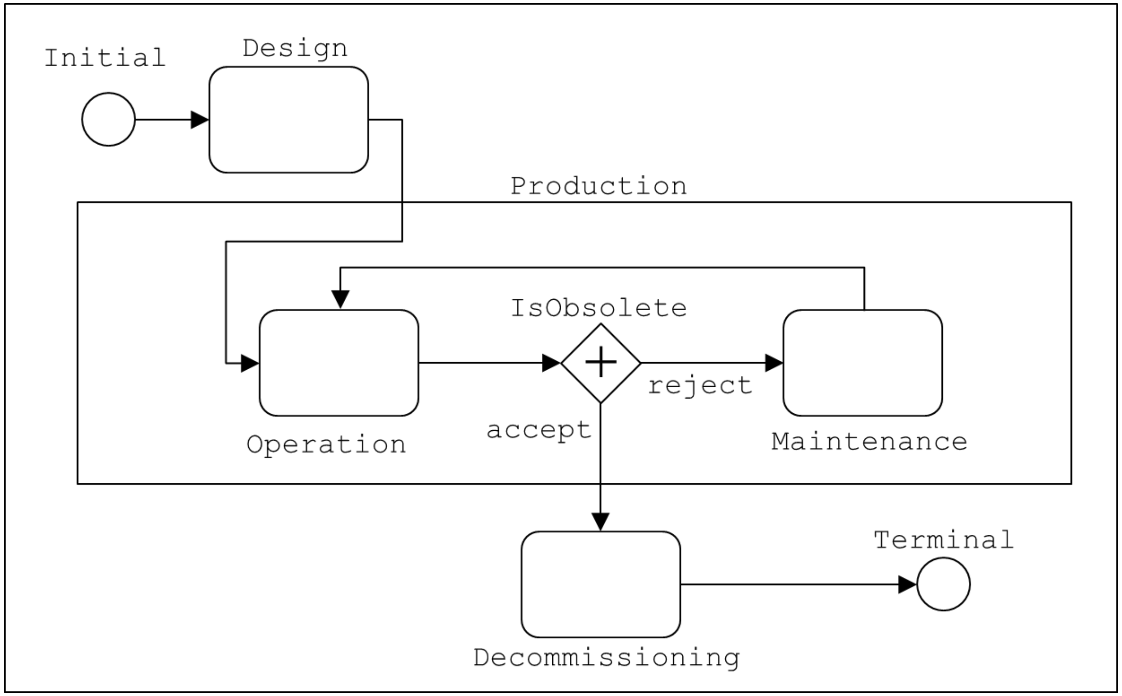
Exercise 4: Travel Reservation

Design a Scola model to represent a travel reservation (flight + hotel)

Life-Cycle (1)


Initial
Design
Production
IsObsolete
reject
accept
Operation
Maintenance
Terminal
Decommissioning

Life-Cycle (2)



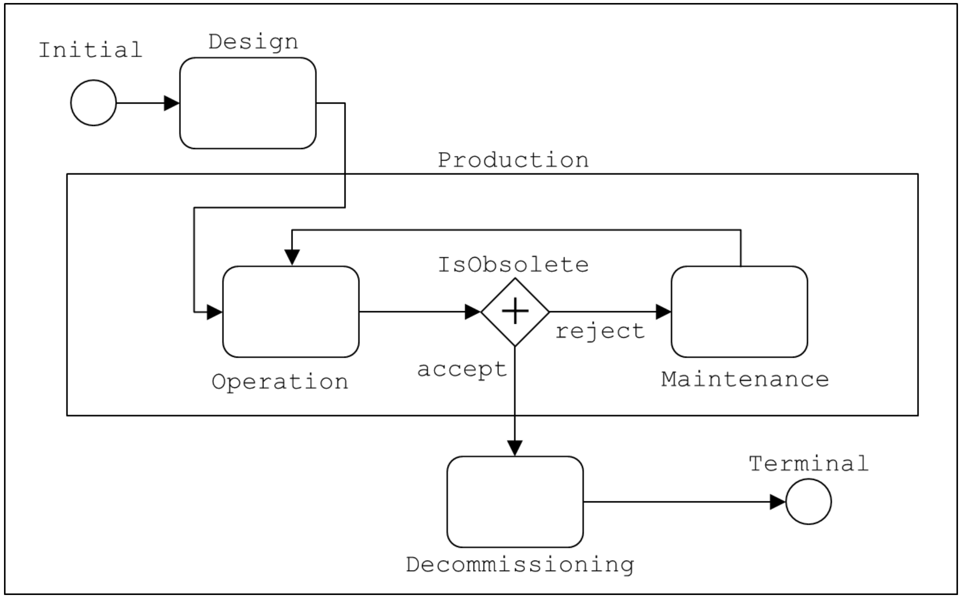
This scenario applies on a system made of a single (implicit) component. It describes the life-cycle of this system.

It involves the sub-scenario `Production` and the choice gateway `IsObsolete`.

Choice gateways describe non-deterministic choices: it is up to the user to tell the simulator which branch to take during the simulation.

Life-Cycle (3)

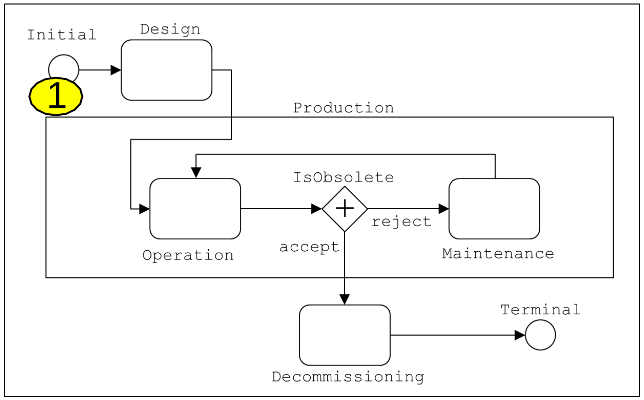
```
scenario LifeCycle
    state Initial
    task Design end
    scenario Production
        task Operation end
        choice IsObsolete
            branch accept
            branch reject
        end
        task Maintenance end
        next Operation IsObsolete
        next IsObsolete.reject Maintenance
        next Maintenance Operation
    end
    task Decommissioning end
    state Terminal
    next Initial Design
    next Design Production.Operation
    next Production.IsObsolete.accept Decommissioning
end
```



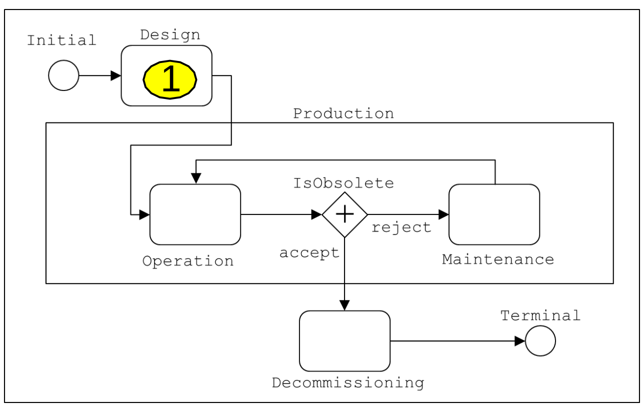
- Sub-scenarios are like macro-states in statecharts

Life-Cycle (4)

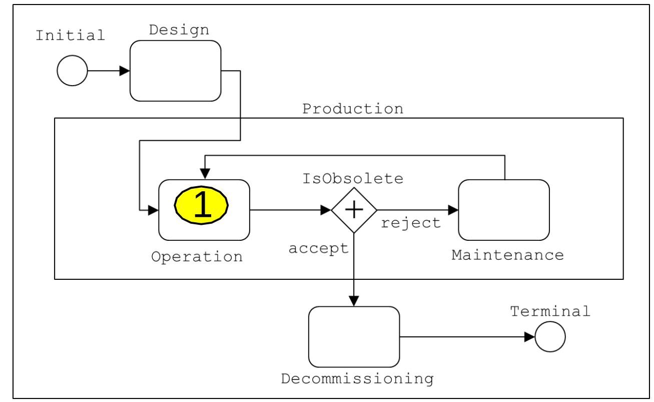
1



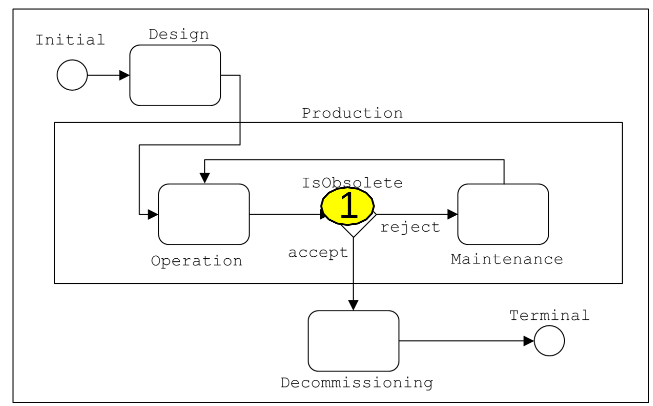
2



3

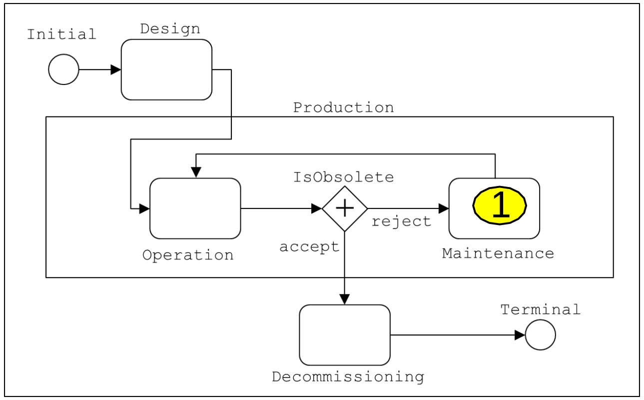


4

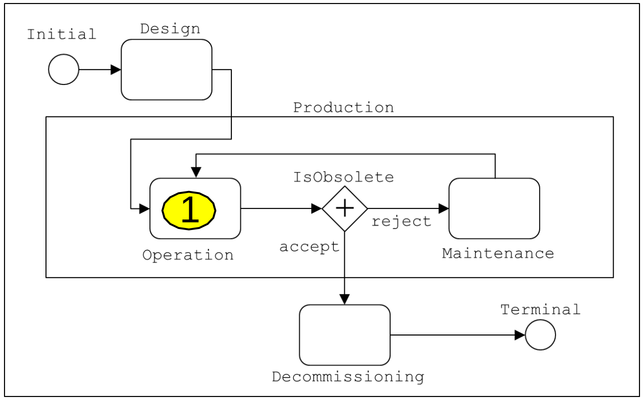


Life-Cycle (5)

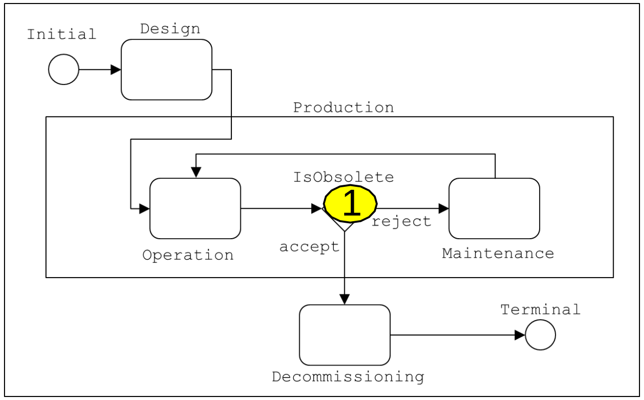
5



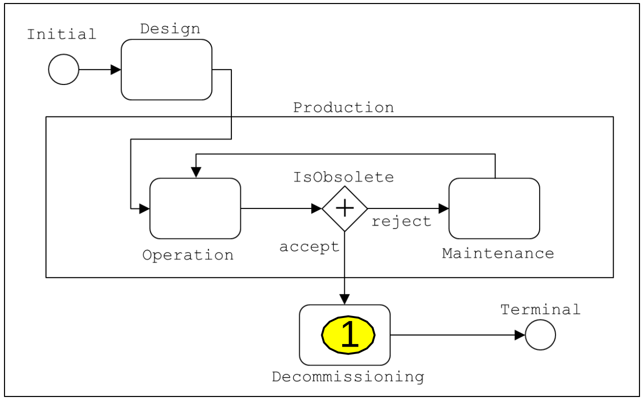
6



7

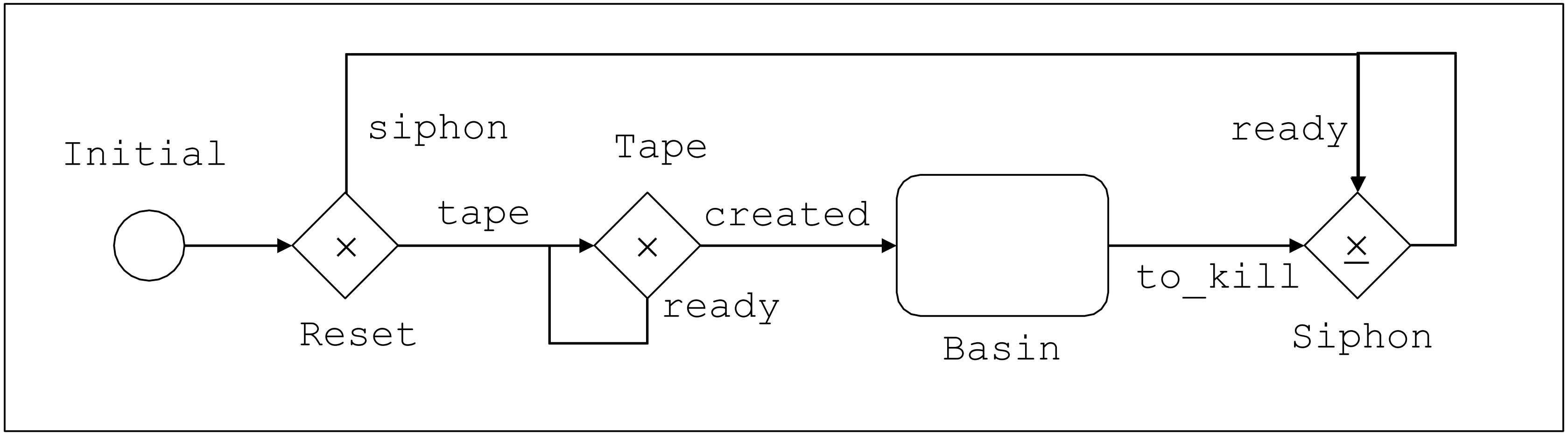


8



and so on…

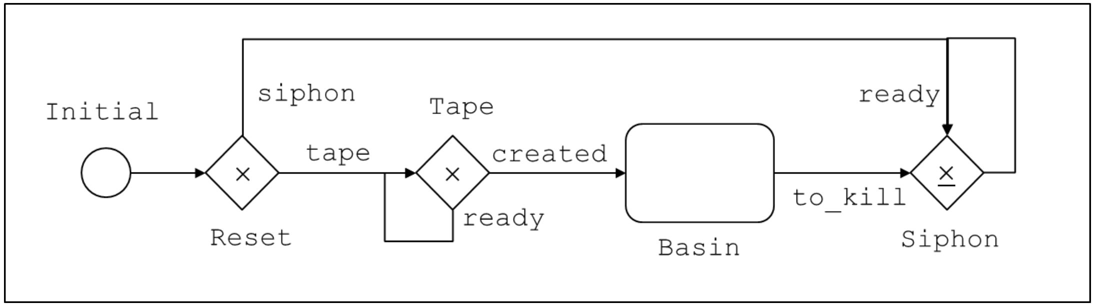
Tapes and Siphons (1)



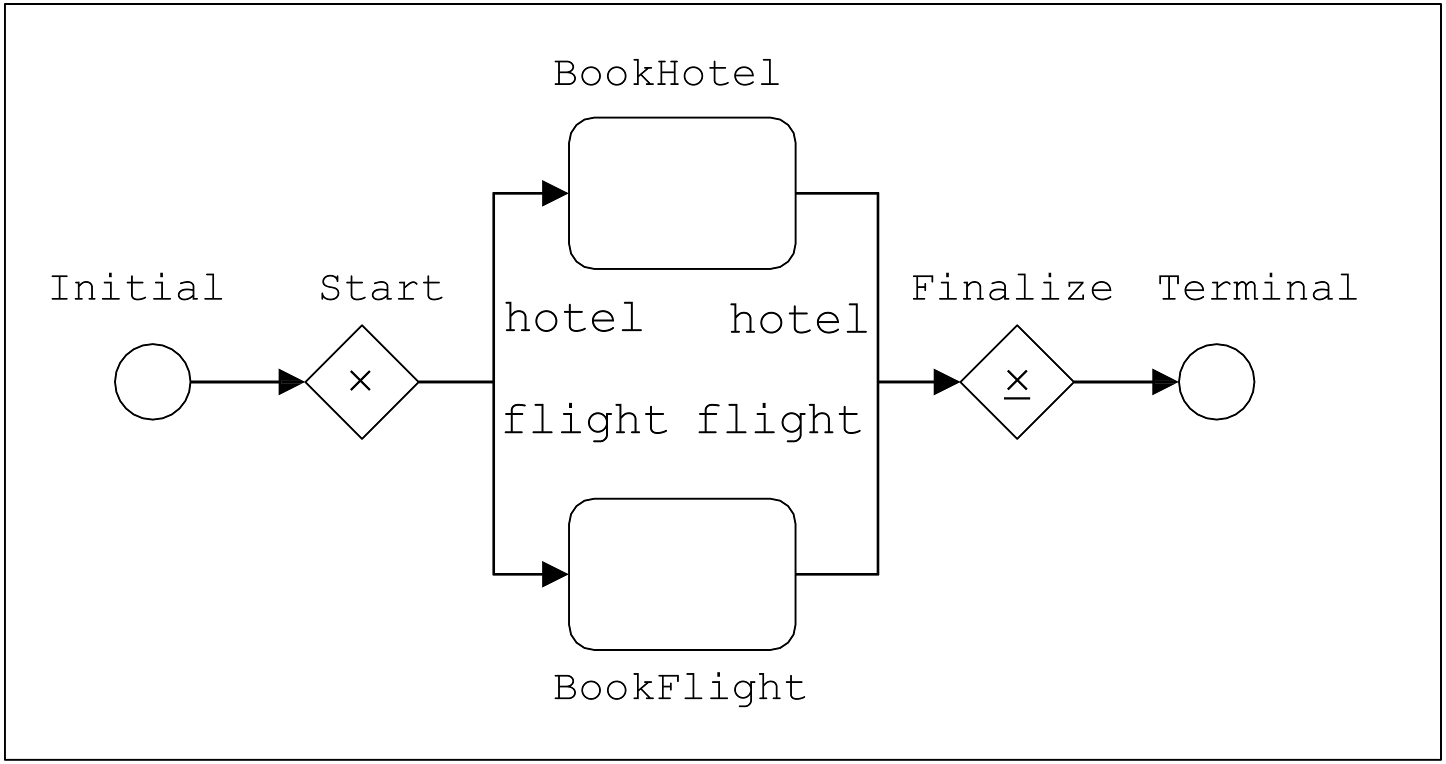
A model that, at the one end, creates as many processes as the analyst wishes (`Tape`) and, at the other end, kills these processes (`Siphon`).

Tapes and Siphons (2)

```
scenario TapeAndSiphon
    state Initial
    fork Reset
        branch tape
        branch siphon
    end
    fork Tape
        branch ready
        branch created
    end
    task Basin end
    join Siphon
        branch ready
        branch to_kill
    end
    next Initial Reset
    next Reset.tape Tape
    next Reset.siphon Siphon.ready
    next Tape.ready Tape
    next Tape.created Basin
    next Basin Siphon.to_kill
    next Siphon Siphon.ready
end
```



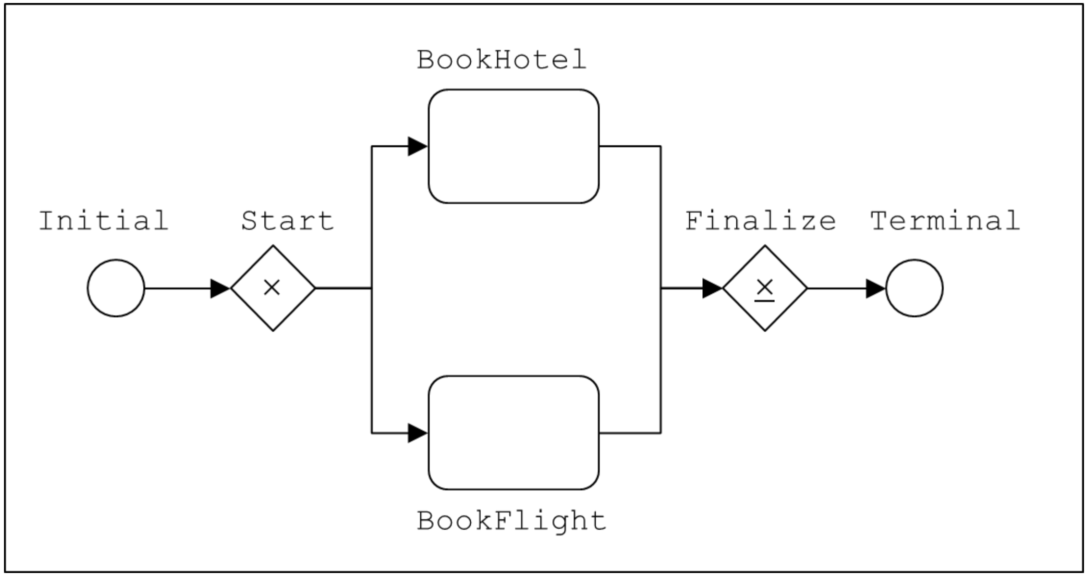
Travel Reservation (1)



In this example, the fork gateway `Start` creates two processes out of one while the join gateway `Finalize` creates one process out of two.

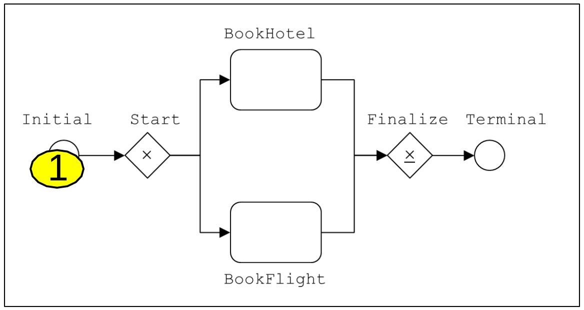
Travel Reservation (2)

```
scenario TravelReservation
    state Initial
    fork Start
        branch hotel
        branch flight
    end
    task BookHotel end
    task BookFlight end
    join Finalize
        branch hotel
        branch flight
    end
    state Terminal
    next Initial Start
    next Start.hotel BookHotel
    next Start.flight BookFlight
    next BookHotel Finalize.hotel
    next BookFlight Finalize.flight
    next Finalize Terminal
end
```

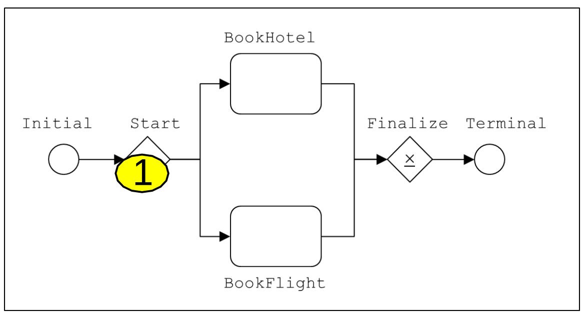


Travel Reservation (3)

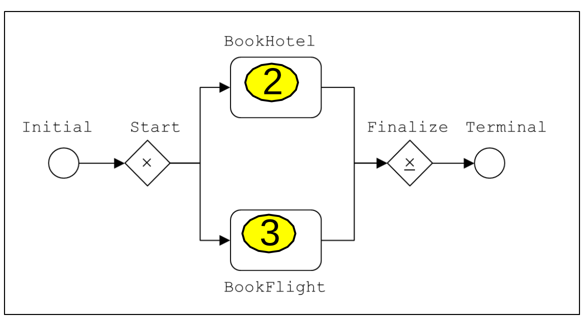
1



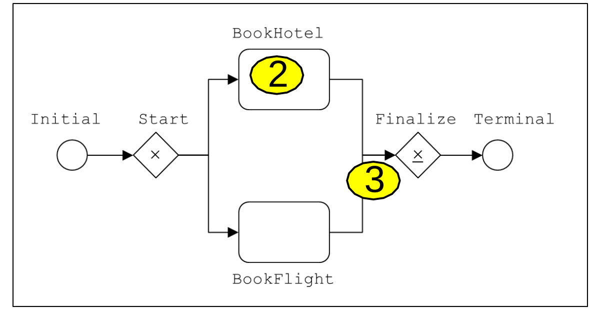
2



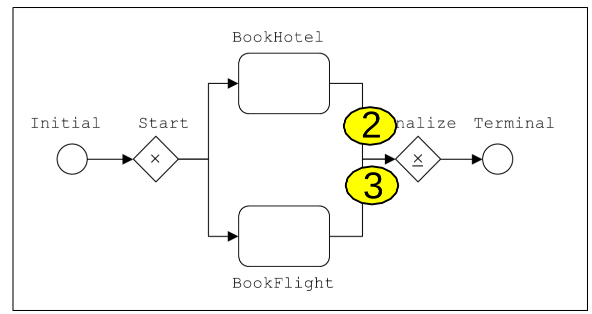
3



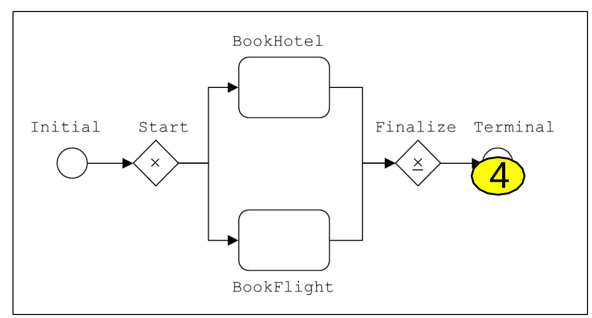
4



5

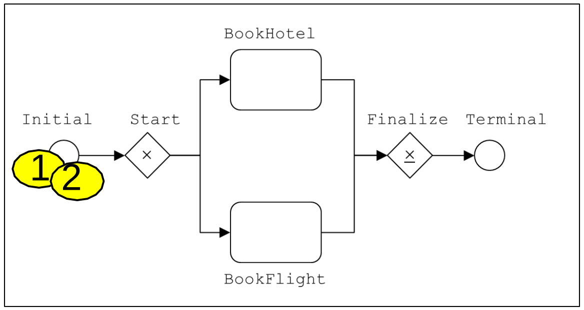


6

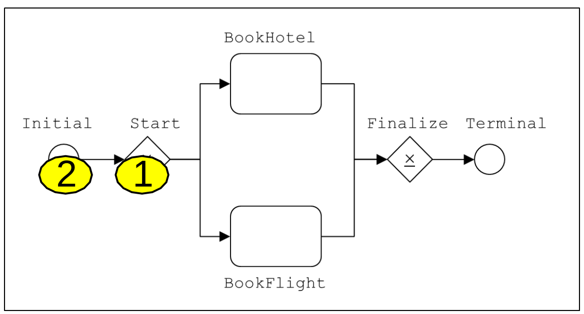


Travel Reservation (4)

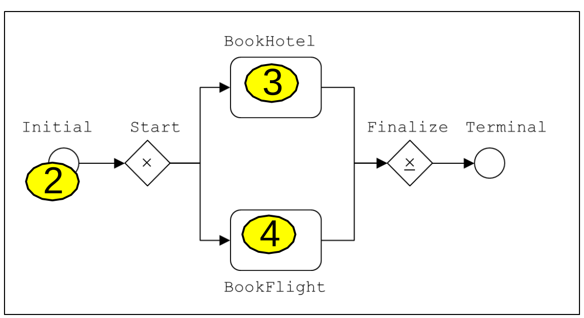
1



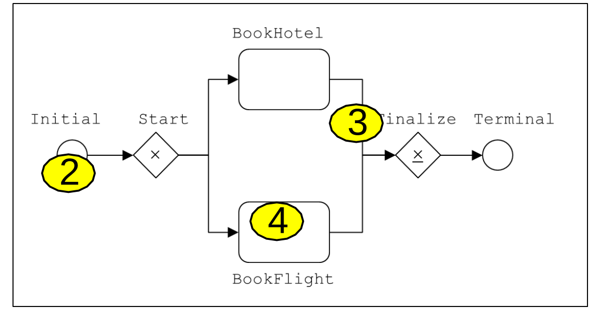
2



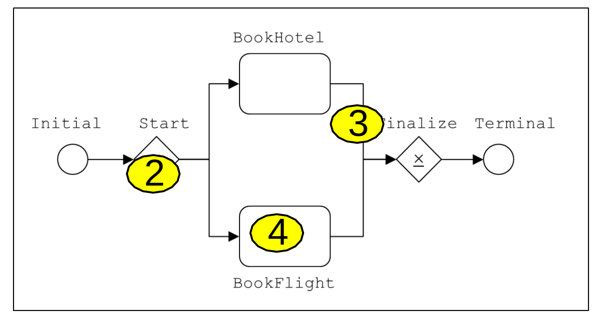
3



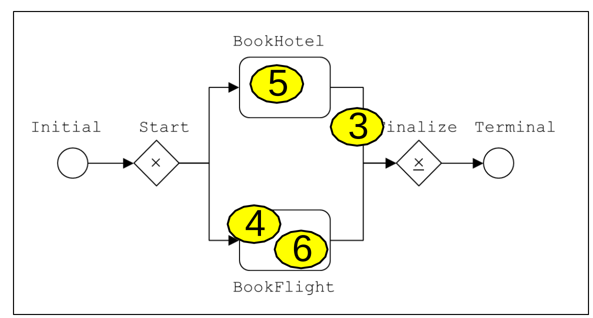
4



5

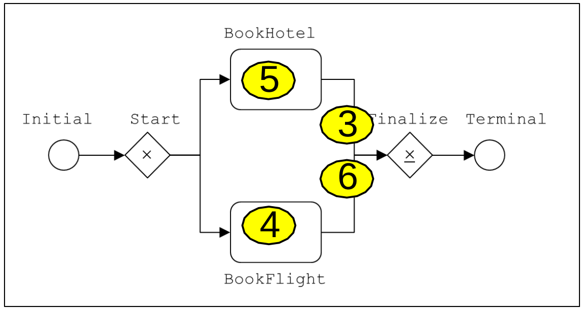


6

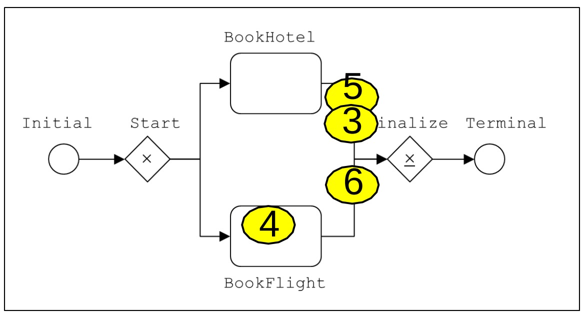


Travel Reservation (5)

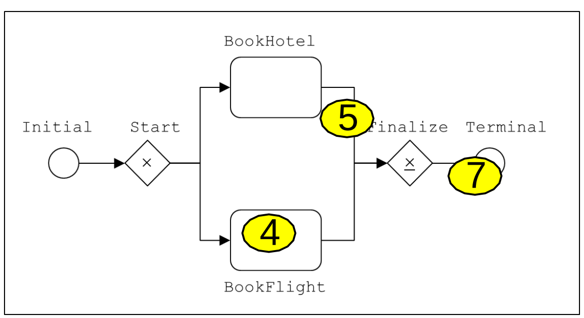
7



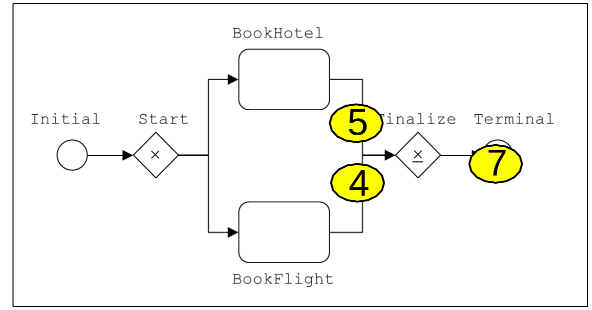
8



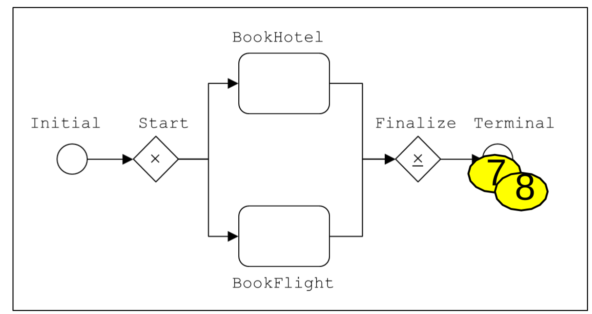
9



1
0



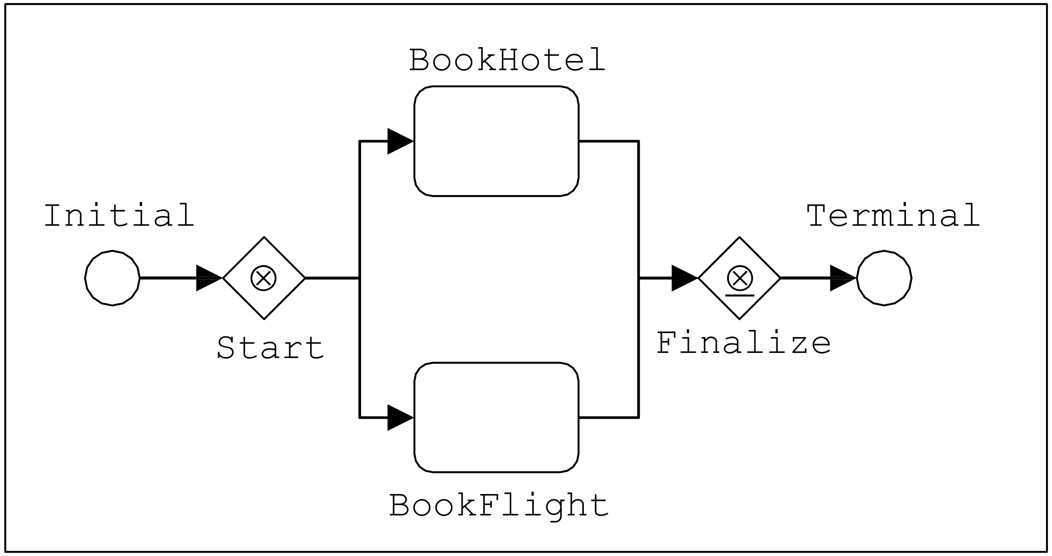
11



Is there a problem?

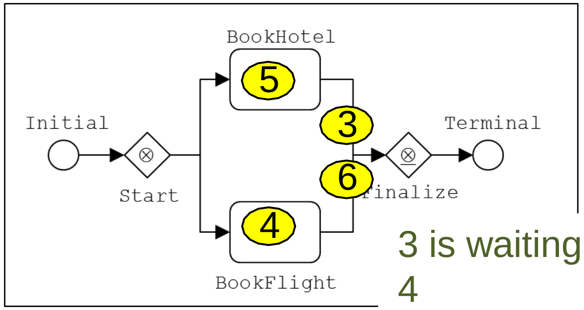
Travel Reservation (6)

```
scenario TravelReservation
    state Initial
    split Start
        branch hotel
        branch flight
    end
    task BookHotel end
    task BookFlight end
    merge Finalize
        branch hotel
        branch flight
    end
    state Terminal
    next Initial Start
    next Start.hotel BookHotel
    next Start.flight BookFlight
    next BookHotel Finalize.hotel
    next BookFlight Finalize.flight
    next Finalize Terminal
end
```



Travel Reservation (7)

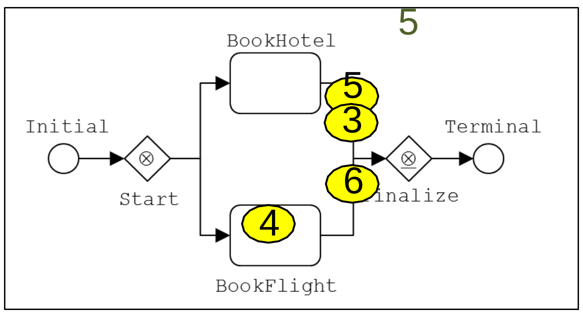
7



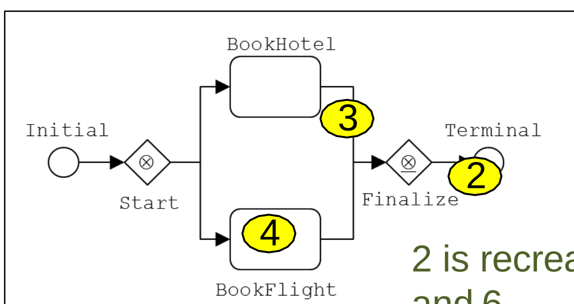
3 is waiting for 4

6 is waiting for 5

8

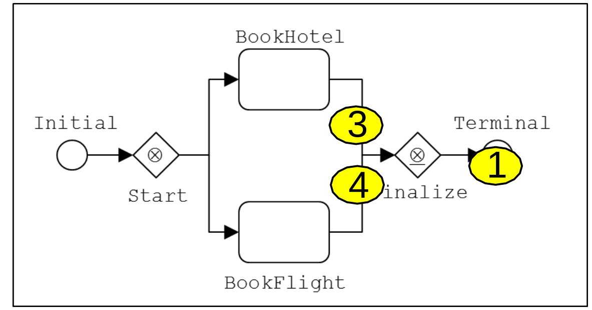


9

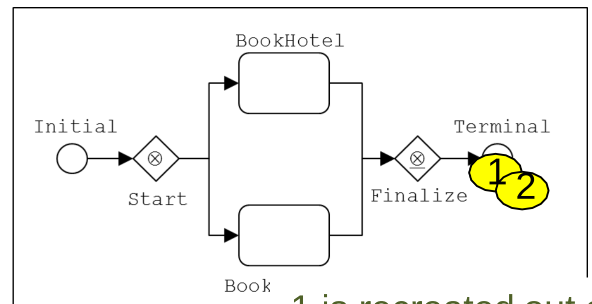


2 is recreated out of 5 and 6

10



11



1 is recreated out of 3 and 4

Exercises (Series 8)

Exercise: Dynamic Car Assembly

Consider a car assembly line. The process is as follows:

- A new car enters into the assembly line.
- It is then moved to a first station where is painted.
- It is then moved to the second station where the engine is assembled.
- It is then move to the third station where the wheels are assembled in two steps: first the front train, then the rear train.
- The car is then delivered (taken out the production line).

Each car must have its own series number.

There can be at most one car at each place of the assembly line, i.e. at the beginning of the line and in each station.

Hint: Use test gateway to prevent a car to be moved to a place where there is already another car. The Boolean expression (is_block *path*) can be used to check the presence of a block at the give place.

Linkk to Dynamic Car Assembly Model: https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/DynamicCarAssembly.pdf

Exercises (Series 9)

<u>Exercise:</u> Largest port

A block `Store` contains an arbitrary number of integer ports. Design a scenario to get the name of the port with the largest value.

Hint: use instruction `if` *`condition`* `then` *`instruction`* and instruction block `begin` *`instructions`* `end`

Linkk to Dynamic Largest Port Model: https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/LargestPort.pdf

Exercises (Series 10)

Exercise 1: Dynamic Car Assembly (revisited)

Design a model that use fail instructions rather than test gateways to solve the dynamic car assembly exercise.

Exercise 2: Master Thesis

Bob is doing his master project under the supervision of Alice. He has to do some research and in parallel to write his master thesis. This requires some iterations with Alice until she gives eventually her approval.

Design a model to represent this process. First, just using ports, without any component creation. Second, with component creation and moving. Third with component creation, sending and reception.

Linkk to Dynamic Car Assembly Revisited Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/DynamicCarAssemblyRevisited.pdf

Linkk to Master Thesis Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/MasterThesis.pdf

Exercises (Series 11)

Exercise 1: Queues

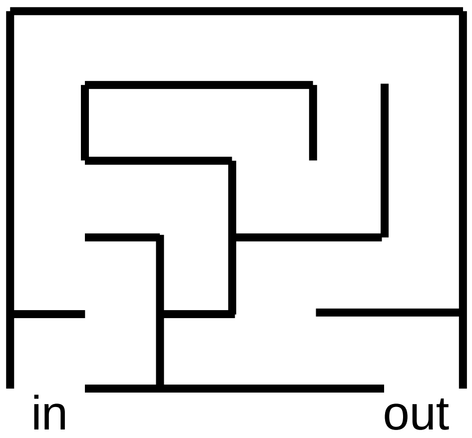
In an shop, clients must choose one of two queues at the cashier. They are served in the order of arrival in the queue they choose.

Design a model for such a system and simulate it.

Hint: use three processes, one to create new clients and one for each queue.

Exercise 2: Maze

Design a Scola model to get out of the following maze.



Hint: recall Tom Thumb.

Linkk to Queues Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Queues.pdf

Linkk to Maze Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Maze.pdf

Exercises (Series 12)

Exercise 1: Eratosthenes

Design a model to calculate prime numbers lower than 100 using Eratosthenes' Sieve.

The idea is to have two nested loops: the outer one to generate candidate numbers (from 3 to 100 in order) and the inner one to test candidates. The test consist in comparing (via a modulo) the candidate with all prime numbers found so far.

Hint: Prime numbers are store as integer ports p1=2, p2=3, p3=5… into a block Primes.

Exercise 2: Ferry

A ferry carries trucks from the left bank to the right bank of a river. It goes forth and back as long as there are trucks to carry. It can contain only one truck at a time. Design a Scola model to represent this ferry.

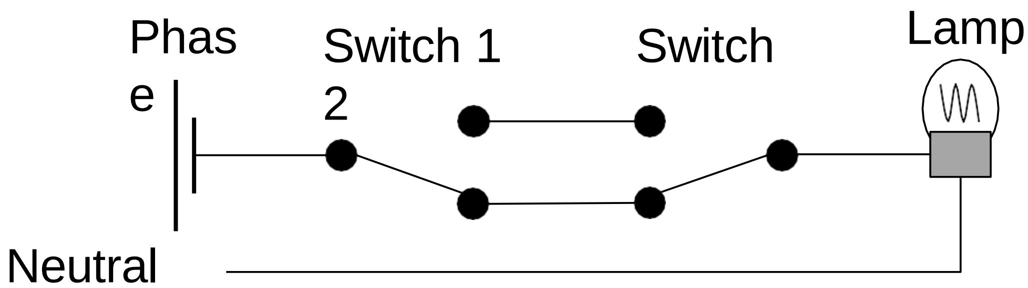
Linkk to Eratosthenes Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Eratosthenes.pdf

Linkk to Ferry Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Ferry.pdf

Exercises (Series 13)

Exercise 1: Two-Way Switch

Modify the code proposed in this section so to model a two-way switch.



Exercise 2: Wages

Alice, Bob and Carol are salespersons. Their monthly wages are calculated as follows.

Fixed salary + 4% of the growth revenue they generate + 800€ if the sum of the two preceding numbers is below 9000€ and 400€ if it above. Design a model to calculate their wages.

Name	Gr. Rev.	Salary	Var. Part	Bonus	Total
Alice	47 500	8 000	1 900	400	10 300
Bob	38 900	6 700	1556	800	9 056
Carol	51 600	9 000	2 064	400	11 464

Linkk to Two-Way Switch Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/TwoWaySwitch.pdf

Linkk to Wages Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Wages.pdf

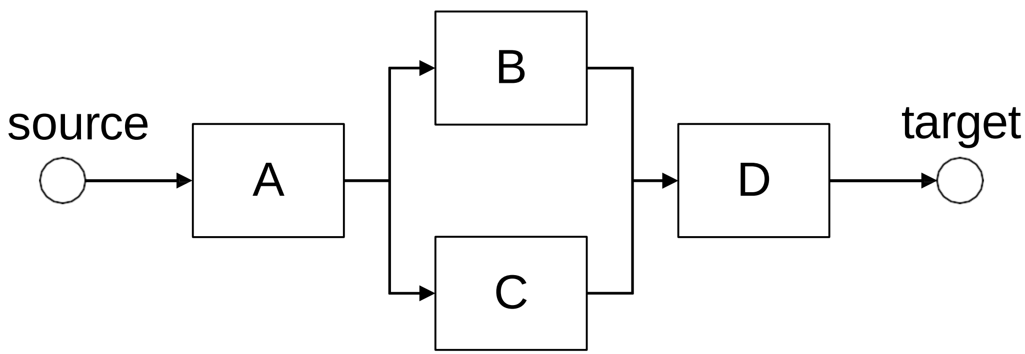
Exercises (Series 14)

Exercise 1: 2-out-of-3 system

A 2-out-of-3 system is a system that works if at least two out of its 3 components are working. Design a model for such a system and simulate it.

Exercise 2: Bridge

Components A, B, C and D of the following reliability block diagram may fail and be repaired. The system described by the diagram is working if there is a working path from the source node to the target node. Design a model for such a system and simulate it.



Linkk to 2-out-of-3 system Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/2OutOf3System.pdf

Linkk to Bridge Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Bridge.pdf

Exercises (Series 15)

Exercise 1: Electric Circuit

Design the complete model of the electric circuit presented in this section. First without cloning nor classes, then with cloning and finally with classes.

Exercise 2: Bridge

Same question with the Bridge exercise of the previous section.

Exercise 3: Collaborative Report

Alice and Bob write a report. Alice makes version 0, then each of them read the report in turn. After reading they can decide either to finalize it, which stops the writing process, or to improve it and to pass it to their colleague.

Design a object-oriented Scola model for this scenario.

Linkk to Electric Circuit (Object Oriented) Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/ElectricCircuitWithClass.pdf

Linkk to Electric Circuit (Prototype Oriented) Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/ElectricCircuitWithCloning.pdf

Linkk to Bridge (Object Oriented) Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/BridgeObjectOriented.pdf

Linkk to Bridge (Prototype Oriented) Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/BridgePrototypeOriented.pdf

Linkk to Collaborative Report Model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/CollaborativeReport.pdf

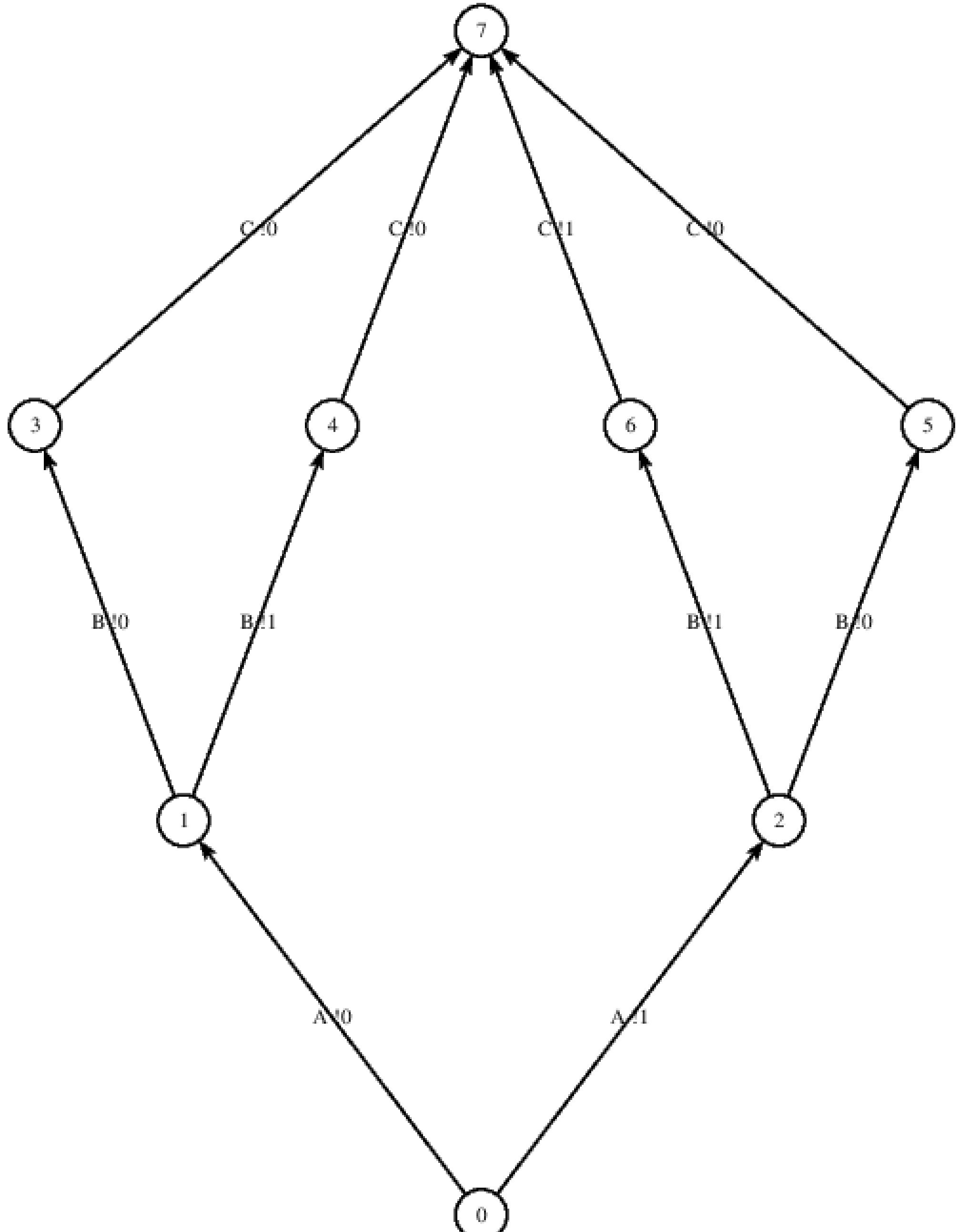
Exercises (Series 16)

Exercise: Given the LOTOS process:

process gate_and = a ?aa:Bit; b ?bb:Bit; c !and(aa, bb); stop

Give its LTS

Give its equivalent SCOLA Model and simulate it


7
C!0
C!0
C!1
C!0
3
4
6
5
B!0
B!1
B!1
B!0
1
2
A!0
A!1
0

a?aa:Bit is equivalent to a!0 [] a!1

```
choice ABIT_Value
		branch Zero
		branch One
end
choice BBIT_Value
		branch Zero
		branch One
end

task AZero set a 0 end
task AOne set a 1 end
task BZero set b 0 end
task BOne set b 1 end

next ABIT_Value.Zero AZero
next ABIT_Value.One Aone
next BBIT_Value.Zero BZero
next BBIT_Value.One BOne

state initial
next initial ABIT_Value
next AZero BBIT_Value
next AOne BBIT_Value
```

Link to Scola model:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/and_gate.pdf

Exercises (Series 17)

Exercise: Write Scola models for these processes:

1. Process1 = a; (b; stop [] c; stop)

1. Process2 = a; b; stop [] a; c; stop

2. Process3 = a; (b; d; stop [] c; stop)

3. Process4 = a; b; d; stop [] a; c; stop

```
/* process1 = a; (b; stop [] c; stop) */
domain Action {NONE, a, b, c, stop} end
block P1
        Action action NONE
end
scenario B as Process1
        task A set action a end
        task B set action b end
        task C set action c end
        task Stop set action stop end
        choice CH
                branch B1
                branch B2
        end
        state S0
        state exit
        next S0 A
        next A CH
        next CH.B1 B
        next B Stop
        next CH.B2 C
        next C Stop
        next Stop exit
end
```

Link:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Process1.pdf

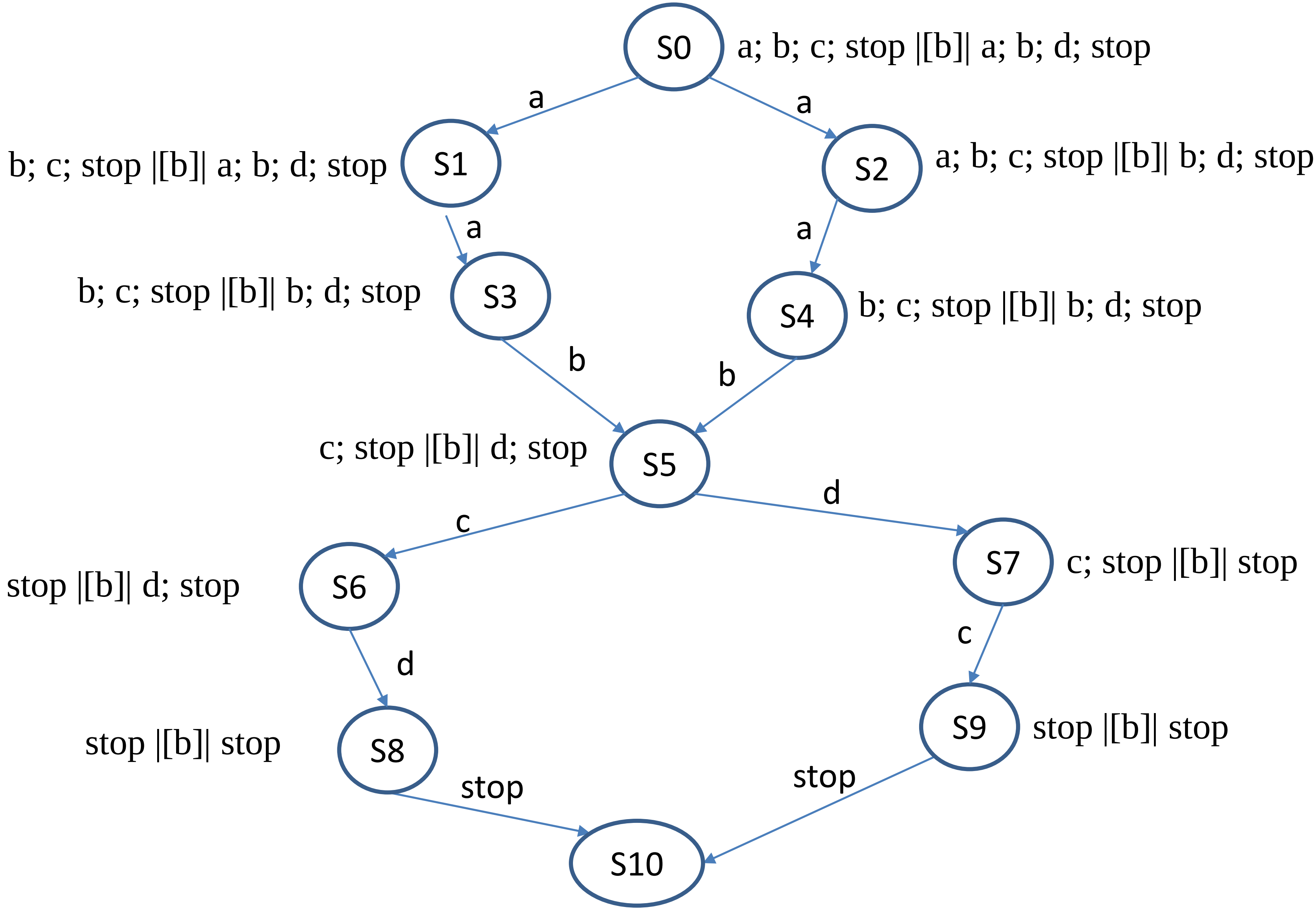
Exercises (Series 18)

Exercise: Give LTS of these processes and then write their Scola models:

Process5 = a; b; c; stop |[b]| a; b; d; stop

Process6 = (a; b; stop) |[b]| (c; b; stop)

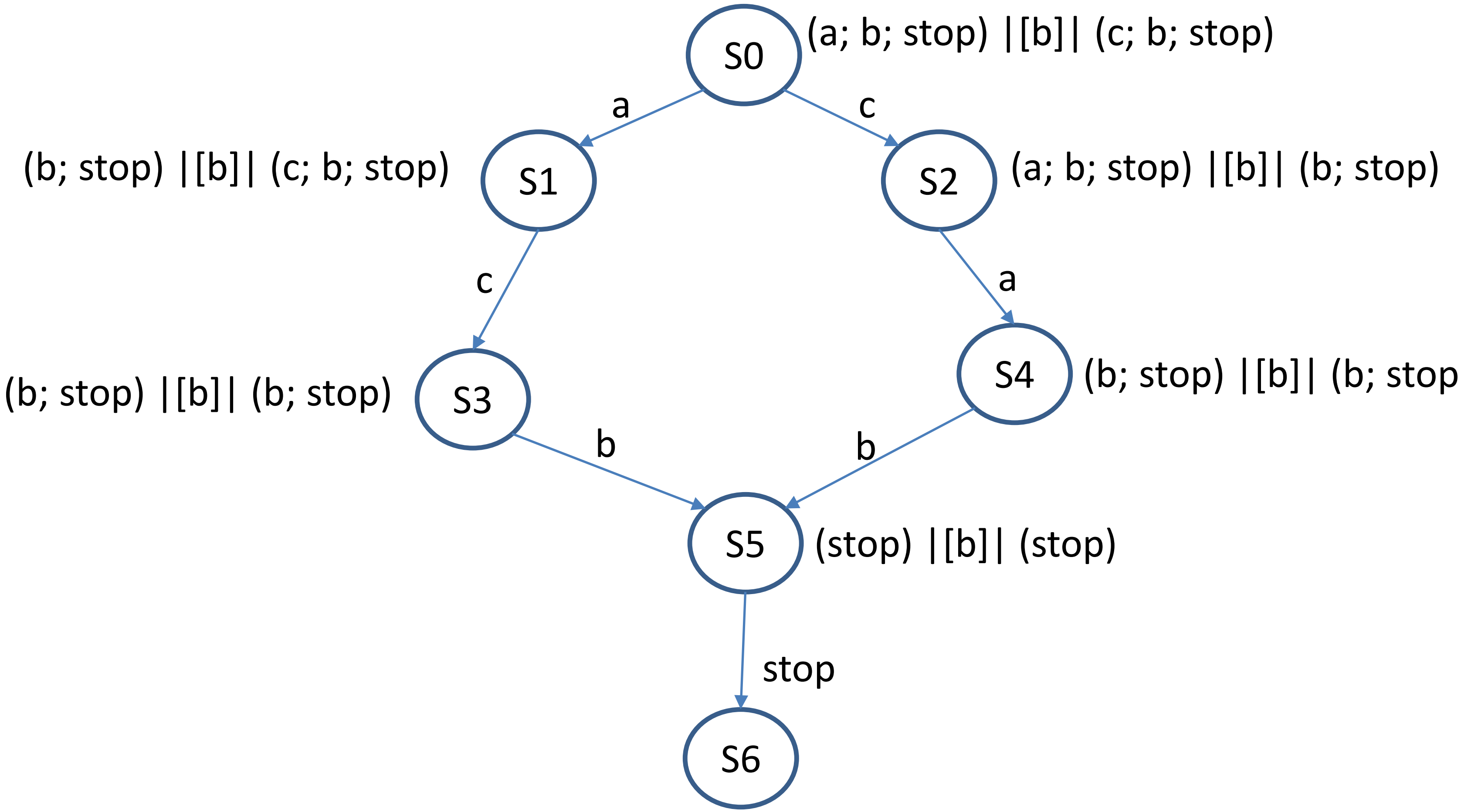
Process7 = Marie et Abdel font séparément le petit déjeuner et le souper, cependant ils déjeunent toujours ensemble: Marie := pd; d; s; stop, Abdel := pd; d; s; stop. Donc Process7 = Marie |[d]| Abdel = (pd; d; s; stop) |[d]| (pd; d; s; stop)



Process5 = (a; a; b [] a; a; b) ; (c; d; stop [] d; c; stop) = a; a; b ; (c; d; stop [] d; c; stop)

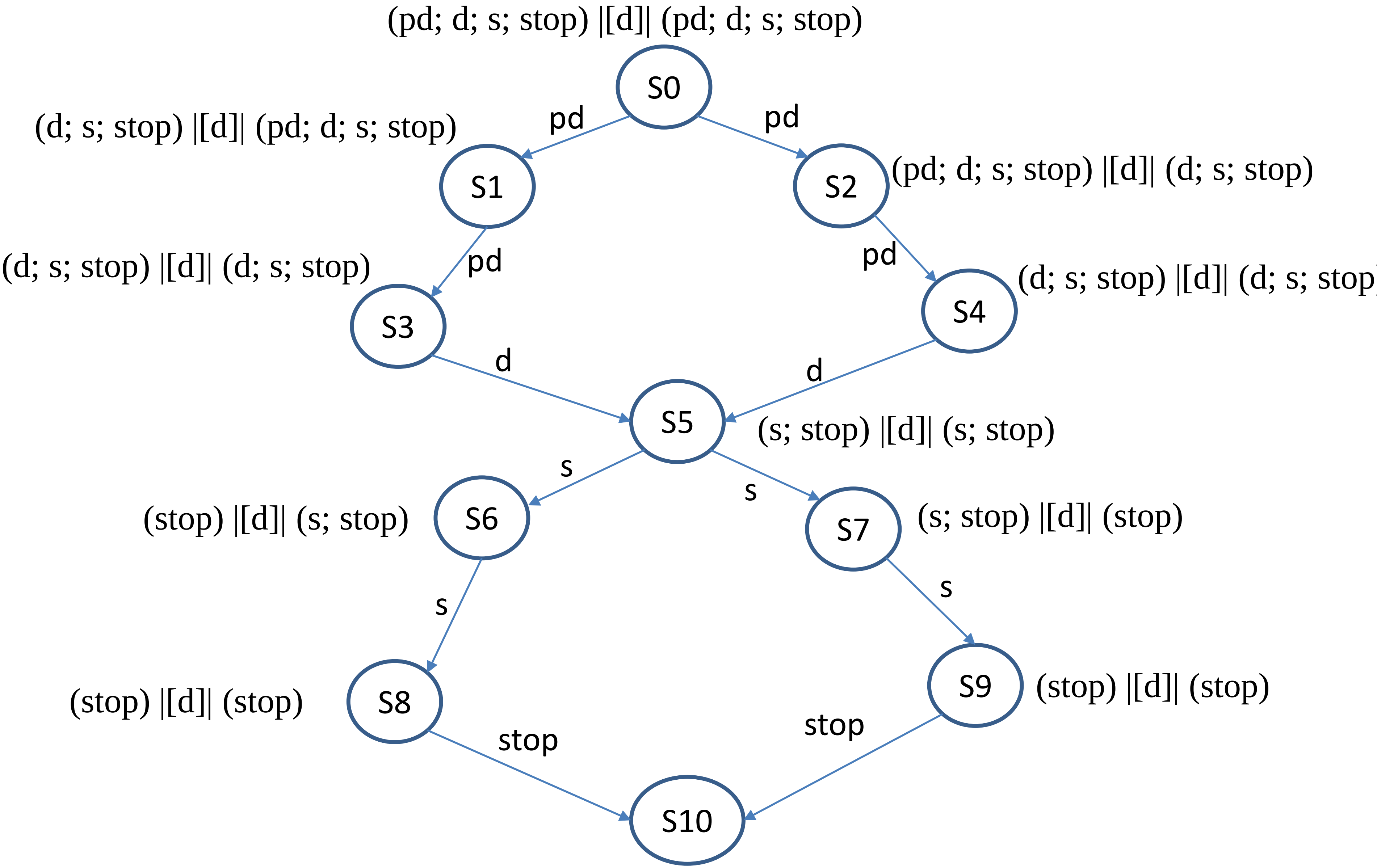
car a [] a = a

Link to Scola model of Process5:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Process5.pdf



Process6 = (a; c; b; stop) [] (c; a; b; stop)

Link to Scola model of process6:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Process6.pdf



Process7 = pd; pd; d; (s; s; stop [] s; s; stop) [] pd; pd; d; (s; s; stop [] s; s; stop) = pd; pd; d; s; s; stop

Link to Scola model of process7:
https://elearning.univ-msila.dz/moodle/pluginfile.php/712399/mod_folder/content/0/ScolaModels/Process7.pdf